Concurrent Controller and Simulator Neural Network Development in the Evolutionary Robotics Process

Grant Warren Woodford

Submitted in fulfilment of the requirements for the degree of Magister Scientiae in the Faculty of Science at the Nelson Mandela Metropolitan University

March 2016

Supervisors: Mathys C. du Plessis & Christiaan J. Pretorius

DECLARATION

I, Grant Warren Woodford (205014224), hereby declare that the dissertation for the degree of Magister Scientiae is my own work and that it has not previously been submitted for assessment or completion of any postgraduate qualification to another University or for another qualification.

_____

Grant Warren Woodford

Acknowledgements

Over the course of my studies, I have received support and encouragement from a great number of individuals. I would like to express my sincere gratitude to my supervisors Mathys du Plessis and Christiaan Pretorius for their continuous support for my studies and providing much feedback, motivation and immense knowledge. I could not have imagined having a better pair of supervisors.

The research presented here would not have been possible without all the years of preliminary research conducted by my supervisors. I would also like to sincerely thank the NMMU Computing Sciences department for providing me an opportunity to join the department as a student and for giving me access to the resources required to successfully complete this research. I have also enjoyed my experiences lecturing one of the modules in the department and providing technical assistance to some of the students.

I thank my fellow students for providing a stimulating, supportive and fun environment that kept me productive over the last two years. In particular, I am grateful to Michael Louwrens for providing the expert advice and technical expertise required for this research to be possible. I would like to thank my family for their support provided over the years.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged (UID number: 89526). Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

Abstract

Evolutionary Robotics (ER) is a field of study that has shown much promise in automating the development of robotic controllers and morphologies. The use of simulators as an alternative to real-world robots is often employed to reduce the time required to develop effective controllers in the ER process. However, the development of adequate simulators is often time-consuming, complex and may require specialised knowledge. Simulators are usually developed before the ER process can begin. Additionally, accurate simulators can be computationally expensive and still be unable to account for all the peculiarities in the robotic system.

Simulators are either constructed from physics models and/or are based on empirically collected data. The vast majority of simulation approaches are based on physics models which can become complex and require special knowledge. Alternative approaches to robot simulation that simplify and automate the modelling of real-world phenomena are therefore important. Some researchers have developed non-traditional approaches to robot simulation, such as using Artificial Neural Networks (ANNs) to model real-world phenomena. Simulators that are constructed from ANNs are called Simulator Neural Networks (SNNs) and are traditionally constructed before the ER process can begin and require the sampling of real-world experimental data. SNNs have good noise tolerance, generalization ability, accuracy, efficiency and simplicity and can be automatically constructed with little specialised knowledge. There are, however, disadvantages to the traditional approach to SNN construction, namely the simulator must be created before the ER process can be initiated and a large amount of behavioural data must be collected in order to accurately predict future behaviours.

Many of the current approaches for improving the simulator development process use bi-directional approaches for simulator development. Bi-directional approaches allow for the simulator to receive feedback from controllers run on real-world robots. The feedback is used for simulator improvement which helps create better controllers

and ideally reduces the gap between the robot's behaviours in simulation and reality. The improvement of the simulator and controllers together reduces the number of robot evaluations required to develop effective controllers. However, bi-directional approaches have mainly been researched for physics-based simulators which have their associated disadvantages.

Due to the advantages that SNNs and bi-directional approaches have both individually shown, and in order to address some of the identified challenges in traditional simulator construction, research presented here proposes and demonstrates that SNNs and controllers can be developed concurrently in a bi-directional approach during the ER process. The proposed approach developed in this research thus benefited from some of the advantages of both SNNs and bi-directional approaches.

In order to prove the viability of the proposed approach, prototypes were developed that successfully demonstrated that SNNs and controllers could be concurrently developed for real-world robots performing trajectory planning tasks. A simple wheeled robot and complex snake-like robot were selected for this work. These robots continually evaluated controllers in the real-world and motion tracking techniques were used to collect behavioural data which was used to train the SNNs. The SNNs were continually trained and used during the ER process to evolve better controllers that were subsequently evaluated on the real-world robots, generating even more relevant behavioural data.

It was found that the approach developed in this research was successfully validated by using the two robot prototypes. Robotic behaviours were adequately simulated by using the developed SNNs which were used to evolve controllers for trajectory planning tasks. This research also conducted a first-time investigation into the viability of using SNNs to simulate a complex snake-like robot.

Influential factors, related to the success of the approach, were studied by investigating various parameter settings of the proposed algorithm. The success of the proposed approach was benchmarked for various settings of the parameters, such

as the controller population size, mutation rate and the tournament selection sizes. Lastly, the scalability, generality and limitations of the approach developed in this research were also investigated.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Term
ANN	Artificial Neural Network
BNS	Bootstrapped Neuro-Simulation
BTR	Back to Reality Algorithm
EA	Evolutionary Algorithm
EC	Evolutionary Computing
EEA	Estimation-Exploration Algorithm
ER	Evolutionary Robotics
GA	Genetic Algorithm
SNN	Simulator Neural Network

Chapter 1

RESEARCH CONTEXT

1.1 Introduction

This research was concerned with the formulation of a new approach for the concurrent development of simulators and controllers using a non-standard simulation approach. Prototypes were developed using the new approach and validated through experimental work. The following section (Section 1.2) gives the background knowledge related the hypothesis given in Section 1.3. The research objectives that focused this work are presented in Section 1.4. Section 1.5 describes the methodology followed in this research and lastly, the dissertation layout is given (Section 1.6).

1.2 Background

Evolutionary Robotics (ER) is a field of study that deals with the automatic, artificial evolution and optimisation of particular traits of autonomous robotic systems [Brooks, 1992; Miglino, Lund, and Nolfi, 1995]. ER is closely related to the field of Evolutionary Computation (EC), as ER utilises many EC concepts. A controller is used to manage the interactions between a robot and its environment [Floreano, Husbands, and Nolfi, 2008]. The ER process requires the evaluation of large numbers of controllers which can be impractical on real-world hardware. Robotic simulators

thus provide a way of speeding up the ER process [Zagal and Ruiz-del Solar, 2007]. A large number of controllers can be evaluated in simulation and thus can evolve to better perform the desired behaviours before being validated on a real-world robot. The development of a simulator is typically required before the ER process can begin. Traditional simulators are often time-consuming and complex to create and can suffer from large disparities between controller behaviour in simulation and reality [Pretorius, 2010]. This can result in ineffective controllers being created by the ER process. Commonly used simulation approaches often require specialised knowledge and the automation of the simulator development process can be difficult.

Many researchers have made great strides in developing and improving traditional simulators for the ER process (Section 2.6). One general approach that stands out is the simultaneous improvement of the simulator and controllers in a bi-directional manner (Section 2.6). During the bi-directional process, controllers are evaluated on a real-world robot and feedback on behaviour is collected to improve the simulator. The simulator is used to evolve controllers for a particular purpose, such as searching for uncertainties in the simulator search space [Bongard and Lipson, 2005] or for accomplishing a particular mission [Mouret, Koos, and Doncieux, 2012]. The improved controllers are periodically selected for real-world evaluations and behavioural data is collected, thus providing more targeted feedback on behaviour for simulator improvement. This allows both the simulator and controllers to improve together in a bi-directional way. This cyclical approach can automate much of the improvement in the effectiveness of the simulator without requiring many real-world controller evaluations. The effort needed to create a comprehensive simulator before the ER process can begin is therefore reduced.

Bi-directional approaches have mainly been applied to physics-based simulators (Section 2.6). However, alternative simulation approaches exist that are simpler to construct and improve. Artificial Neural Networks (ANNs) can be used as an alternative to a physics-based simulator to predict robot behaviours (Section 2.5), referred to as Simulator Neural Networks (SNNs). SNNs have the same disadvantages

as any empirically constructed simulator, such as the collection of large amounts of experimental data or the loss of generality where small changes in the robotic system may require the development of a completely new simulator [Pretorius, 2010]. No known work has investigated the viability of combining a bi-directional and SNN approach.

1.3 Hypothesis

This research hypothesises that viable controllers and simulators can be concurrently created by using bi-directional principles and SNNs. Controllers are evolved for a particular purpose using ER. In order to minimize the specialised knowledge and human intervention required for the simulator development process, SNNs are used for the simulation approach. It is proposed that a novel technique for the concurrent development of controllers and SNNs is possible and such an approach would be advantageous in that there would be no need to develop a simulator before the ER process. Since the simulator is constructed during the ER process, potentially fewer real-world evaluations would be required and disparities between reality and simulation could be reduced.

1.4 Research Objectives

The following research objectives were identified:

- Develop a new approach for the concurrent creation of SNNs and controllers in the ER process.
- Identify and test the various factors pertinent to the success of the proposed approach.
- Consider the scalability and generality of the approach by assessing different robot morphologies.

- Compare the effectiveness of the new approach for tasks of varying levels of complexity.
- Consider the limitations of the approach and identify future potential.

1.5 Methodology

Research related to SNNs and bi-directional approaches is investigated in Chapter 2. A review of related work will aid in the identification of what needs to be considered for the proposal of a new approach to controller and simulator development. This work proposes a real-time controller and simulator development procedure where the chosen simulation approach uses SNNs (Chapter 3). Prototypes were developed of the proposed approach and were validated using real-world robots (Chapters 4 and 5). The prototypes were validated by using two different robot morphologies and by creating controllers for specific tasks in simulation and observing how well behaviours developed in simulation transfer to the real-world robot. The validation experiments also determined the influential factors and limitations of the approach developed in this research by conducting a large number of experiments completely in simulation for various parameter settings. Lastly, conclusions are drawn and the future potential of this work are discussed (Chapter 6).

1.6 Dissertation Layout

The dissertation layout is illustrated in Figure 1.1. This diagram illustrates the different chapters contained in this dissertation. Chapters 2 and 3 present the background to the problem statement, the proposed approach is introduced and motivations for the approach are discussed. The experimental method used to prove the viability of the proposed approach is also presented. Chapters 4 and 5 then discuss the experimental work conducted in order to validate the viability of the proposed

approach. Finally, conclusions are drawn in Chapter 6. The chapters will address the following issues:

- **Chapter 2**: Related work that is applicable to this study is presented. Theory relating to the ER process is discussed and existing studies related to robotic simulators are presented. Various bi-directional approaches that can be applied to simulator development are reviewed. The use of SNNs as an alternative to physics-based simulators is discussed.

- **Chapter 3**: Motivations for the need to develop a new approach for the development of controllers and SNNs are discussed. The proposed approach developed as part of this research is presented and the experimental method used to prove the viability of this approach is discussed.

- **Chapter 4**: This chapter covers the experimental work performed on the first experimental robot (a differentially-steered robot). The procedures used to investigate the proposed approach are discussed and details of the experiments are presented. Results of the experimental work are presented and discussed.

- **Chapter 5**: The experimental work related to the second experimental robot (a snake-like robot) is presented. In order to prove the viability of the proposed approach to scale to a more complex robot, experimental work was conducted using the proposed approach on the snake-like robot. The results of the experimental work are presented and discussed.

- **Chapter 6**: An overview of the experimental work is discussed and outcomes of the research objectives are presented. The contributions of this work are mentioned and limitations of the proposed approach are discussed. Lastly, recommendations for future research are also proposed.

- **Appendix A**: The conference paper that was accepted and presented at the 2015 IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2015) is given.

- **Appendix B**: Presents the paper accepted and published in the Robotics and Autonomous Systems (affiliated with the Intelligent Autonomous Systems (IAS) Society) journal.

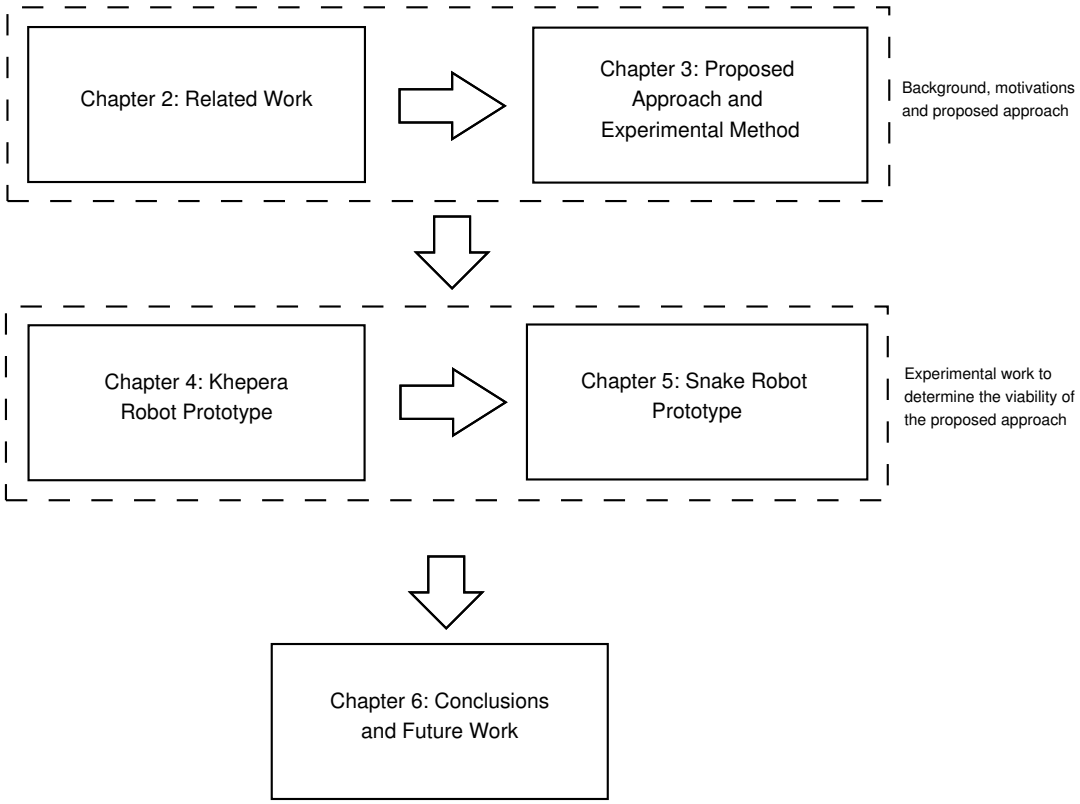


Figure 1.1: Structure diagram for the dissertation

Chapter 2

RELATED WORK

2.1 Introduction

The following section discusses some of the important theory behind ANNs (Section 2.2). Evolutionary Computing (EC) techniques are presented and related to Evolutionary Robotics (ER) in Section 2.3. The importance of simulators in robotics and the ER process are discussed in Section 2.4, while SNNs are described in Section 2.5. The existing work related to bi-directional approaches in ER is presented in Section 2.6 which motivates why this work focuses on developing a new bi-directional approach. Conclusions of this chapter are given in Section 2.7.

2.2 Artificial Neural Networks

The ability of the human brain to perform tasks such as perception and motor control has prompted research into the algorithmic modelling of biological neural systems which has resulted in the creation of ANNs [Engelbrecht, 2007]. ANNs have been shown to outperform conventional technologies in areas such as pattern recognition, certain types of data processing, data-mining, real-time adaptive control of robots and the modelling of many complex phenomena [Maren, Harston, and Pap, 2014].

The following sections describe the creation and functioning of Artificial Neurons

(Section 2.2.1) and how these neurons can be combined to form ANNs (Section 2.2.2). Lastly, methods for training ANNs are addressed in Section 2.2.3.

2.2.1 The Artificial Neuron

ANNs are composed of computational units called Artificial Neurons (ANs) (Figure 2.1) that are connected by directed links. A single AN takes as input a vector of numeric signals $(x_1, x_2, ..., x_n)$ and returns a single numerical output y.

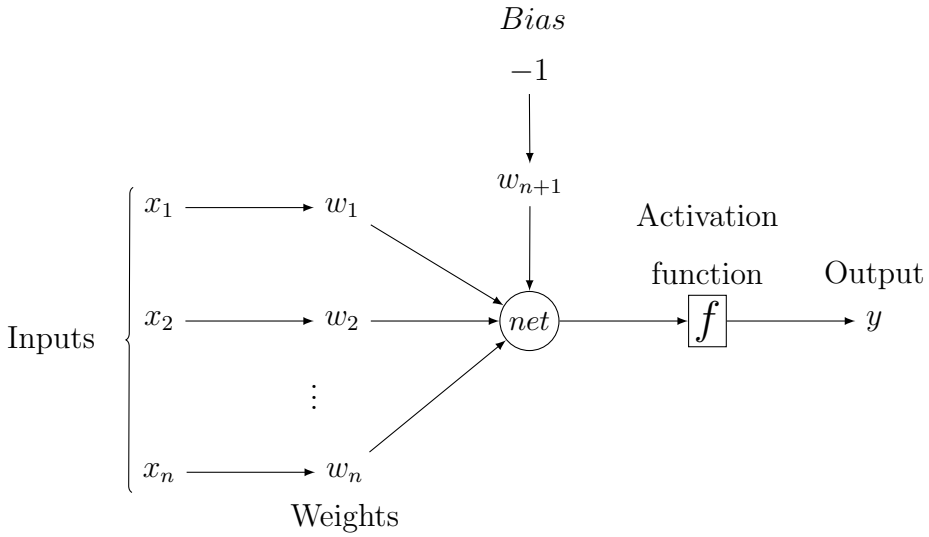


Figure 2.1: Artificial Neuron

For the AN illustrated in Figure 2.1, each weight w_i is associated with an input signal x_i (for $i = 1, 2, 3, ..., n$). Each weight strengthens or weakens its associated input. The *net* input of an AN is typically calculated as the weighted sum of all input signals (Equation (2.1)) where n represents the number of input signals. Every input x_i and weight w_i are real numbers. The input node x_{n+1} is called the bias and typically has a value of -1 which allows for the more effective representation of certain functions [Engelbrecht, 2007]. The inputs propagate through the neuron connections towards the neuron. The strength of the propagation depends on the given inputs and the associated weights of the connections.

$$net = \sum_{i=1}^{n+1} x_i w_i \tag{2.1}$$

The computed *net* signal of the AN is used as the input for the activation function. The resulting output of the activation function is returned as the output of the AN. Activation functions are important for scaling and manipulating inputs and to ensure that an entire network of unit nodes can model non-linear functions [Russell and Norvig, 2010]. There are different types of activation functions that can be employed and most are monotonically increasing such that the range of outputs is either,

$$f : \mathbb{R} \to [0, 1] \tag{2.2}$$

or

$$f : \mathbb{R} \to [-1, 1] \tag{2.3}$$

An exception is the linear activation function which is monotonically increasing on a range $[-\infty, \infty]$. The monotonically increasing property of activation functions is required for certain types of ANN training algorithms. The linear, step, ramp and sigmoid activation functions are described below and illustrated in Figure 2.2.

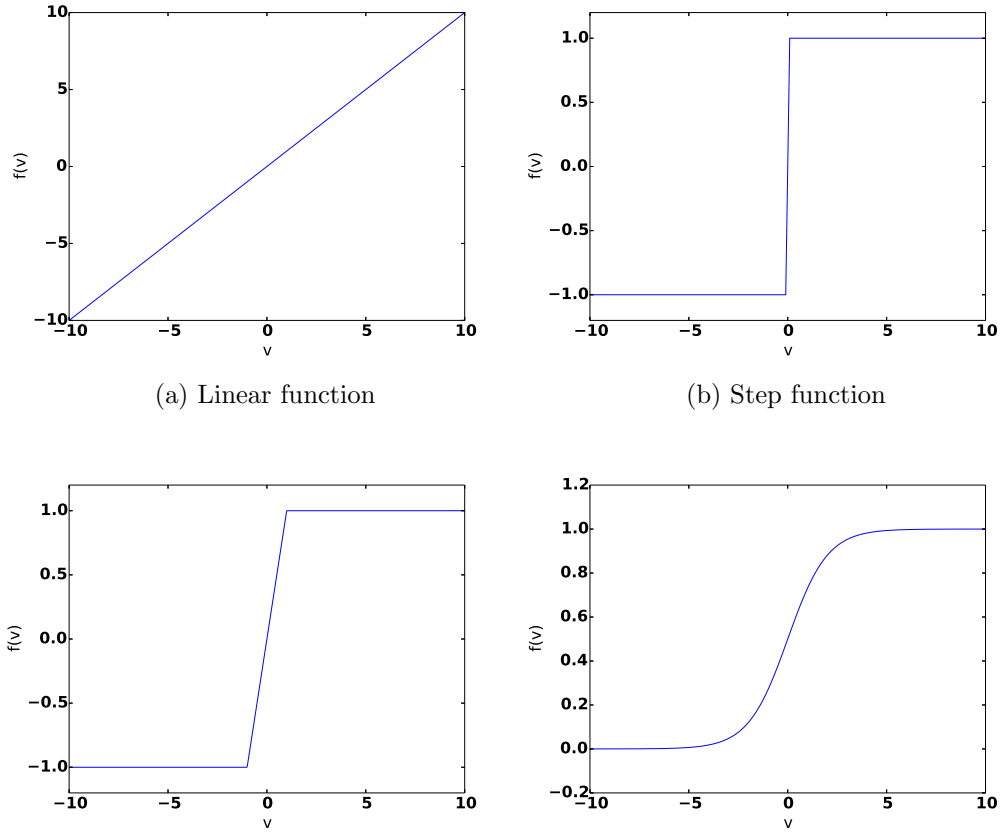
A commonly used function is the linear activation function (Figure 2.2a) and is given by equation (2.4) where m is the slope and v is the computed net signal.

$$f(v) = mv \tag{2.4}$$

The step function (Figure 2.2b) maps inputs to one of two scalar output values which are commonly required for binary classification schemes. Commonly used output values for the step function are -1 and 1. The step activation function is given in equation (2.5).

$$f(v) = \begin{cases} -1 & \text{if } v < 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.5}$$

The ramp activation function makes use of a high and low threshold. This function is illustrated in Figure 2.2c and given in equation (2.6). A fixed value is returned for inputs above the high threshold and another value is returned for inputs below the low threshold. The ramp function acts similarly to the linear function between the high and low thresholds.



(a) Linear function

(b) Step function

(c) Ramp function

(d) Sigmoid function

Figure 2.2: Activation functions

$$f(v) = \begin{cases} -1 & \text{if } v \leq -1 \\ v & \text{if } -1 < v < 1 \\ 1 & \text{otherwise} \end{cases} \tag{2.6}$$

Another commonly used activation function is the sigmoid function (Figure 2.2d) which is given by equation (2.7), where λ is the steepness of the curve.

$$f(v) = \frac{1}{1 + e^{-\lambda(v)}} \tag{2.7}$$

The following section discusses how Artificial Neurons can be networked together to form Artificial Neural Networks (ANNs).

2.2.2 Neural Networks

An ANN consists of one or more interconnected groups of ANs. There are various multilayer ANN topologies that have been developed. Certain applications require that different ANN topologies be used. The Feed-Forward Neural Network (FFNN) is a common topology and is illustrated in Figure 2.3. Each circular dot represents a single AN and the arrows represent connections between ANs. The ANs are commonly grouped into different functional layers. A FFNN typically consists of three layers, namely the input layer, hidden layer and the output layer. There is typically only a single hidden layer, but the number of hidden layers can be zero or more. The hidden layer enables the FFNN to more accurately model complex functions [Engelbrecht, 2007].

The signals propagate through each layer from the input layer, through the hidden layer and lastly to the output layer. FFNNs do not have any feedback connections to previous layers. Neurons in the input layer take as input values $z_1, ..., z_i, ..., z_I$ and the bias input z_{I+1}. These input values are multiplied by their associated weights and the results are used as the input values for the hidden layer neurons $y_1, ..., y_j, ..., y_J$. The hidden layer neurons calculate their respective *net*

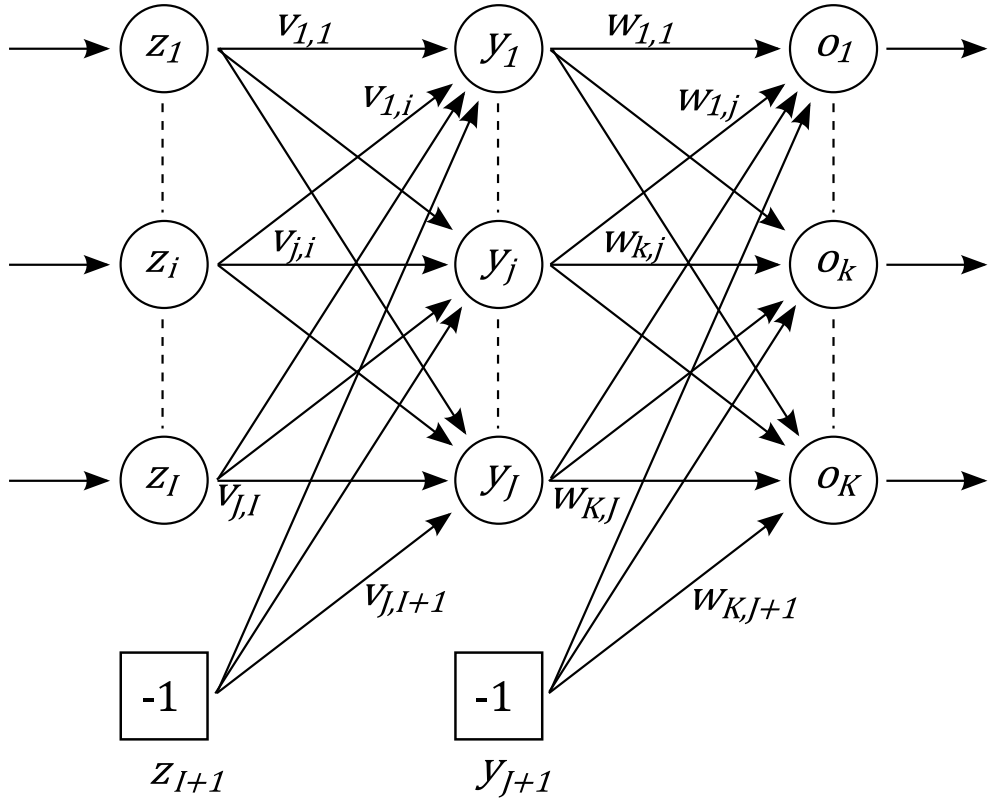


Figure 2.3: A Feed-Forward Neural Network [Engelbrecht, 2007]

(Equation (2.1)) results and one of the activation functions mentioned in the previous section is applied. The outputs from the hidden layer neurons are used as input values for the output layer neurons $o_1, ..., o_k, ..., o_K$. The *net* results of these inputs are calculated for each output neuron and the activation functions are applied. The results are returned as the final output of the FFNN.

Given a specific input, the output for neuron o_k is calculated as shown in equation (2.8) [Engelbrecht, 2007].

$$o_k = f_{o_k}(\sum_{j=1}^{J+1} w_{k,j} f_{y_j}(\sum_{i=1}^{I+1} v_{j,i} z_i)); \quad 1 \leq k \leq K \tag{2.8}$$

where f_{o_k} and f_{y_j} are the activation functions of the neurons o_k and y_j, respectively. I and J are the number of input and hidden neurons respectively.

2.2.3 ANN Training

The output produced by an ANN is dependent on the inputs used and the strengths of the weights connecting the neurons. Weights are optimised by using a training process to allow the ANN to approximate a function based on the training data set. The training set consists of input-output pairs. Each one of these input and ideal output pairs is called a training pattern. Another set of data, referred to as the validation set, is commonly used to test the generalization ability and accuracy of the trained ANN. The validation set is not presented to the ANN during training. Supervised learning algorithms can be used to optimize ANN weights by reducing the errors between the ANN's predicted outputs and the training data set's ideal outputs. The standard measure of the error between predicted and ideal outputs is called the mean squared error (MSE) [Engelbrecht, 2007]. A common algorithm for minimizing the MSE is called Backpropagation which makes use of Gradient Descent optimization techniques [Russell and Norvig, 2010].

The MSE is calculated using equation (2.9) [Engelbrecht, 2007].

$$MSE = \frac{1}{N}\sum_{p=1}^{N}\sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2 \tag{2.9}$$

where N is the total number of patterns in the training set and K is the number of neurons in the output layer. The terms $t_{k,p}$ and $o_{k,p}$ represent the ideal and predicted outputs respectively for the k^{th} output neuron and the p^{th} training pattern in the training set. Once training is complete, the ANN is able to approximate the outputs of arbitrary inputs that were not presented during the training process.

A problem that can occur during the training process is called over-fitting [Russell and Norvig, 2010]. Over-fitting occurs when the error between the ANN's predicted output and the training set's ideal output is driven to a very low value, however, patterns not presented during training have a large error. This is because the ANN is over-trained to memorize the training set and is not trained to generalize for predictions on unseen inputs. Over-fitting can be avoided by observing the errors

of a validation set during the training process and by stopping before over-fitting occurs.

2.3 Evolutionary Computation

Evolutionary Computation (EC) is a field within Computational Intelligence that is well-suited to finding adequate solutions to continuous and discontinuous optimisation problems [Engelbrecht, 2007]. In EC, Evolutionary Algorithms (EA) are inspired by natural evolution, whereby survival of the fittest is the main goal. Survival is achieved through reproduction. Individuals with weak characteristics die off while fitter individuals are more likely to reproduce and hopefully pass on those characteristics that aid in the survival of their offspring. The Evolutionary Robotics (ER) process makes use of the Darwinian principle of survival of the fittest that can be accomplished by using EAs for the automatic creation of robotic controllers [Floreano *et al.*, 2008].

2.3.1 Evolutionary Algorithms

Evolutionary Algorithms (EA) represent a collection of EC paradigms that make use of evolution to perform a stochastic search through the solution space to determine the optimal solution to a specific problem [Engelbrecht, 2007]. The pseudo-code of a generic EA is given in Algorithm 1 [Engelbrecht, 2007].

The EA initializes a population consisting of randomly generated, individual solutions. Each solution consists of a number of genes which represent encoded traits that determine behaviour. A fitness function is a performance measure of how well an individual can perform a given task, relative to other individuals. If the fitness function is not properly designed, the EA might not converge on a solution at all or could converge on an inappropriate solution [Beasley, Martin, and Bull, 1993].

When every individual in the population has been assigned a fitness value, the

Algorithm 1: The Evolutionary Algorithm

Let generation $t = 0$

Initialize a population, $C(0)$, of n individuals

while *stopping condition(s) not true* **do**

  Evaluate the fitness of each individual in $C(t)$

  Create offspring through reproduction operators

  Use selection operators to create a new population $C(t + 1)$

  Advance to the next generation: $t = t + 1$

end

EA produces offspring through reproduction. Individuals of the next generation are sourced either through cross-over operations between parents or directly from the parents. The cross-over process between parents only takes place with a certain probability, known as the cross-over rate [Engelbrecht, 2007]. Parents are chosen for reproduction based on selection operators that are biased towards more fit individuals.

Upon completion of the reproduction process, the offspring can undergo a mutation process. Mutations introduce small, random changes to the genes of individuals which increases the diversity of the population [Beasley *et al.*, 1993]. To ensure that fit individuals are not mutated to the point of distortion, mutations are usually small and only take place with a certain probability, called the mutation rate [Engelbrecht, 2007]. Particular implementations, where mutations are large during the early stages of the EA, but decrease exponentially as the fitness of individuals converge, are known to improve convergence speed and accuracy [Engelbrecht, 2007]. The above process is iterated for many generations, ideally leading to a highly optimised solution.

Possible stopping conditions could be a limit on the number of generations executed or could place a limit on the number of fitness evaluations. These limits should be large enough to allow adequate exploration of the solution space [Engelbrecht,

2007]. A convergence criterion that determines if the population has converged can also be used together with other stopping conditions to determine when to terminate the EA. Other examples of convergence criteria are: when no improvement is observed over a certain number of generations, if there is little change to the individuals of a population over consecutive generations, or if an acceptable solution has been found [Engelbrecht, 2007].

2.3.2 Evolutionary Robotics

Controllers manage the interactions between the robot and its environment. Sensors and actuators of the robot provide input data to the controller, which in turn produces appropriate output responses, such as motor movements [Miglino *et al.*, 1995]. Controllers have been developed by using techniques such as Genetic Algorithms [Pratihar, 2003], Evolutionary Programming [Koza, 1992] and Evolutionary Strategies [Schwefel, 1993].

The focus need not lie exclusively with the creation and evolution of controllers. Robot morphologies and controllers can be evolved in a simulator, thus allowing optimizations of both the controller and the morphology according to some desired behaviour [Lund, 2003].

ER seeks to automate the development of robot controllers through the use of EAs, as opposed to creating controllers manually. This is important because as robots, environments and tasks become more complex, the manual creation of controllers becomes less feasible [Sofge, Potter, Bugajska, and Schultz, 2003].

Figure 2.4 illustrates the ER process applied to an ANN-based controller. A population of individuals is initialized by randomly generating them. All subsequent generations of individuals are created by using selection operators (**A**). Each individual consists of a set of encoded characteristics which form an individual's genes (**B**). Each set of characteristics is actually a representation of an ANN-based controller that can be used to manage a robot's interaction with its environment (**C**).

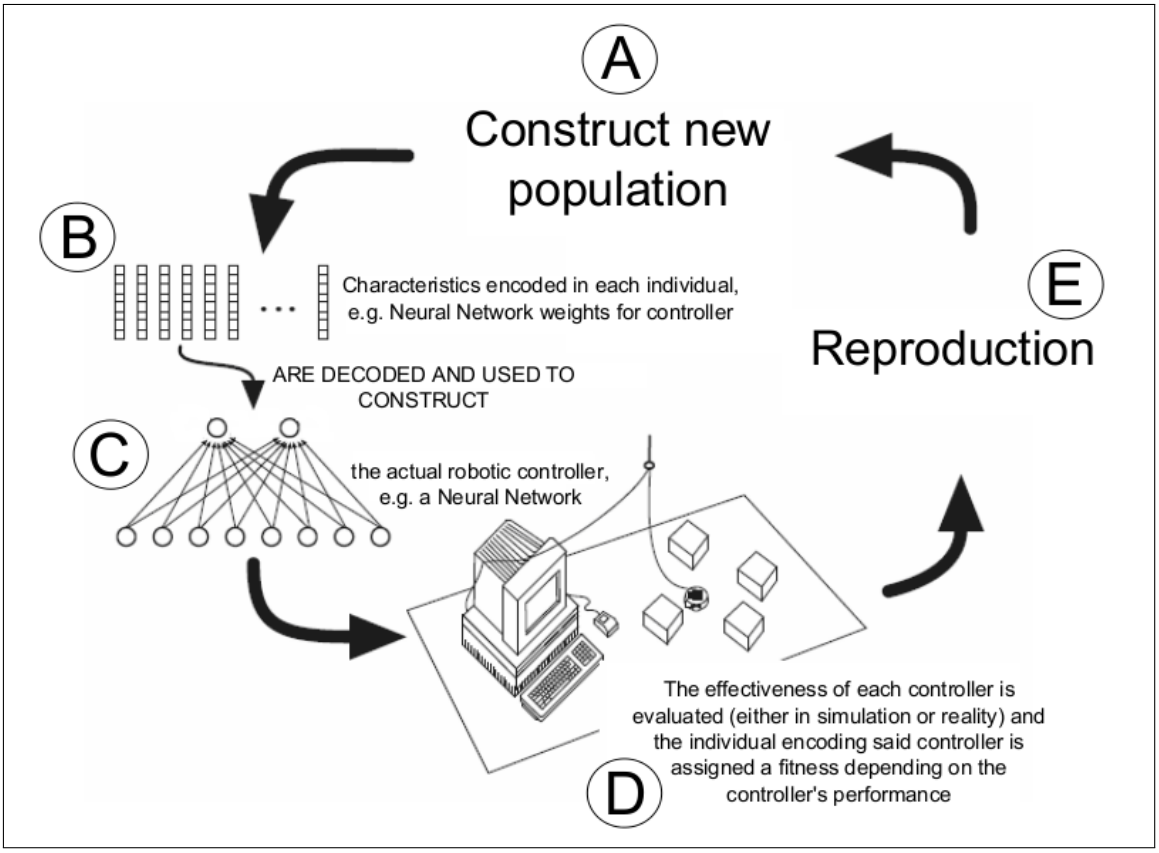


Figure 2.4: The Evolutionary Robotics Process applied to the evolution of an ANN-based controller [Floreano *et al.*, 2008; Pretorius, 2010]

The population of controllers is evaluated to determine how well each individual performs a given task. The fitness evaluations are done by transferring controllers to a real-world robotic system or an approximation is used by means of a computer simulation (**D**). Once the population of controllers has been assigned fitness values, the next step is the creation of a new population of controllers (a new generation) by using the cross-over and mutation operators (**E**). The above process is repeated for many generations until some stopping condition is met. An ANN-based controller was used in the above mentioned ER process but, any type of controller could be utilized.

2.4 Simulators

Evaluating controllers on real-world robots is time-consuming and can damage hardware through mechanical wear [Floreano and Mondada, 1994; Zagal and Ruiz-del Solar, 2007]. The evaluation of poorly performing controllers can especially damage robot hardware [Floreano *et al.*, 2008], because of erratic robotic movements during the early stages of evolution. Real-world controller evaluations may require frequent manual interventions [Floreano *et al.*, 2008], such as repositioning the robot or restoring the initial environment conditions. Manual interventions may not be feasible for the large number of evaluations required in the ER process. The time taken to perform hardware evaluations increases with the complexity of the robotic system, but there are often time constraints in performing these evaluations [Zagal and Ruiz-del Solar, 2007].

Simulators provide a way to overcome the issues inherent in hardware evaluations. Simulators allow controllers to explore the search space faster than would be possible using real-world hardware, by reducing the number of physical evaluations required [Lund and Miglino, 1996].

Much research in ER is concerned with overcoming the challenges in using simulators effectively [Bongard, 2013]. Challenges in simulator design are inaccuracies and/or oversimplification in the modelling of certain phenomena. Oversimplified or inaccurate simulators may result in controllers that rely too heavily on peculiarities that exist only in simulation, but are not present when the controller is evaluated in reality, commonly referred to as the *reality-gap* problem [Jakobi, Husbands, and Harvey, 1995]. Oversimplification can be avoided by using highly accurate simulators. However, even highly accurate simulators cannot model reality perfectly and will inevitably contain inaccuracies [Floreano *et al.*, 2008]. Highly accurate simulators can also be computationally expensive [Miglino, Nafasi, and Taylor, 1994]. Scalability can become an issue where the time taken to evaluate controllers can grow substantially with increased complexity in the robotic system [Bongard, 2013;

Matarić and Cliff, 1996]. Thus, simulators ideally need to provide highly accurate representations of reality whilst not being too computationally expensive to operate. Simulators need to compromise between providing a highly accurate representation of reality and computational efficiency to allow for the evaluation of many candidate controllers [Pretorius, 2010].

Simulators can be broadly categorized into three classes, namely physics-based simulators, empirical models and hybrid models [Pretorius, 2010]. Physics-based simulators use physics models to describe the interaction between the robot and its environment. Empirical models are simulators that are created from experimentally collected data. Hybrid models are a combination of both physics and empirical models.

Simulators can be categorized as either using a behaviour-based representation or a knowledge-based representation, with various intervening levels in the modelling of the underlying robotic system [De Nardi, 2010]. A knowledge-based representation entails the principled formulation of the different components of the underlying system, resulting in clearly defined structures and parameters that have physical interpretations. An example of one of these components is the modelling of the friction between the robot and the operating surface [Echeverria, Lassabe, Degroote, and Lemaignan, 2011]. Behaviour-based representations are focused on modelling the underlying behaviours of the robotic system based on experimentally collected data [De Nardi, 2010]. For example, the behaviour representations are less interested in modelling frictional properties that form part of the robot's behaviour and more interested in predicting the robot's actual behaviours.

An advantage of simply aiming to model behaviours is that simulator creation often requires little in-depth knowledge of the underlying system, however, the models usually do not provide meaningful real-world interpretations [De Nardi, 2010]. Certain implementations of behaviour-based models can provide meaningful physical structures such as Genetic Programming [De Nardi, 2010; Schmidt and Lipson, 2009].

Knowledge-based representations have the advantage of providing a clear structure of the model. Models contain parameters that usually have real-world interpretations which are useful for being able to understand and explain the underlying systems. However, knowledge-based models may become complex due to a large number of possible components of the system and the inability to accurately model certain phenomena.

For any given simulation approach, the model taxonomy can also be described based on the amount of prior knowledge built into the model [De Nardi, 2010]. The term white-box refers to models that are perfectly known and are constructed entirely from prior knowledge and physical insight. Models that are based, in part, on some form of specialised knowledge and empirical data are known as grey-box. Models that do not make use of physical insights or prior knowledge of the underlying system are black-box. An example of a black-box model is when ANNs are used to model particular phenomena [De Nardi, 2010].

The Open Dynamics Engine (ODE) [Smith, 2007] is a popular simulator that uses a physics-based engine. Physics-based simulators, like ODE, make use of rigid body dynamics, Newtonian mechanics and collision detection algorithms to simulate reality [Laue, Spiess, and Röfer, 2006]. It has been shown that physics-based simulators are able to effectively evolve controllers that are able to retain their evolved behaviour when transferred to reality [Bongard and Lipson, 2004; Jakobi *et al.*, 1995; Moeckel, Perov, Nguyen, Vespignani, Bonardi, Pouya, Sproewitz, van den Kieboom, Wilhelm, and Ijspeert, 2013]. However, physics-based simulators cannot perfectly simulate real-world systems [Bongard, 2013; Floreano and Mondada, 1994]. This is because of the inherent difficulty in taking into account all the physical aspects present, such as weight, friction and inertia. Highly accurate physics-based simulators are however possible [Carpin, Stoyanov, Nevatia, Lewis, and Wang, 2006], but the design of such systems may require specialised knowledge about complex physics models and environments [Wittmeier, Jäntsch, Dalamagkidis, and Knoll, 2011]. The development and use of accurate simulators can be financially costly, as additional

expertise may need to be acquired or ER researchers will need to invest much of their time acquiring such expertise [Floreano and Mondada, 1994].

A technique called *minimal simulation* has been shown to reduce the complexities inherent in simulating real-world phenomena [Jakobi, 1998]. This approach only tries to model those aspects in the robotic system that are relevant for developing the required behaviours. All aspects that are not modelled are randomly varied such that evolving controllers only exploit those phenomena that have been modelled. This can result in computationally efficient simulators that can approximate complex environments well enough to evolve controllers that exhibit the required behaviours in reality [Floreano *et al.*, 2008]. A potential disadvantage to minimal simulations is that it may be difficult to identify which aspects of the system are relevant. Specialised knowledge may be required to identify those parts of the system that need to be simulated.

An alternative approach to the use of a complex simulator is the use of a simplified model for controller evaluations. The developed behaviours will likely not transfer well to reality, however, reinforcement learning or evolutionary techniques can then be applied in the real-world. This approach has been effective in reducing the number of real-world evaluations required to develop certain behaviours [Abbeel, Quigley, and Ng, 2006].

The use of empirical models as simulators has been proposed [Lund and Miglino, 1996; Nolfi and Parisi, 1995]. Models are constructed from data sampled from real-world experiments which can be used to simulate robotic sensors [Miglino *et al.*, 1995]. This results in the creation of black-box models where no prior knowledge about the dynamics governing the robotic system is built into the models. Empirical models can often be more effective than using physics models because empirical models are better able to capture the fuzzy characteristics of real-world sensors [Lund and Miglino, 1996].

Empirical models allow for the use of various data-mining methods to be used in the simulator development process. For example, experimental data can be divided

into several clusters and the data in each cluster can be modelled separately [Kamio and Iba, 2004]. The use of multiple simulators to develop behaviours in simulation can reduce the occurrence of solutions that exploit inaccuracies or peculiarities that exist in simulation but are not present in reality [Togelius, De Nardi, Marques, Newcombe, Lucas, and Holland, 2007].

Many methods of constructing empirical models use primitive data analysis techniques, such as interpolation or lookup tables [Lund and Miglino, 1996]. Empirical models can, however, be created by using more advanced techniques, such as NARMAX polynomials [Nehmzow, Kerr, and Billings, 2009] or Gaussian Processes [Lizotte, Wang, Bowling, and Schuurmans, 2007]. Genetic Programming techniques have been shown to be particularly effective in the automatic generation of symbolic equations that explain real-world phenomena from experimental data [De Nardi and Holland, 2008; Schmidt and Lipson, 2009]. The notion of using ANN-based simulators (SNNs) to model real-world phenomena has also been proposed and successfully applied by some researchers [Lee, Nehmzow, and Hubbold, 1998, 1999; Nakamura and Hashimoto, 2007; Pretorius, du Plessis, and Cilliers, 2013; Togelius *et al.*, 2007]. Empirical modelling techniques such as Genetic Programming and SNNs are able to automate the construction of simulators without the need for specialised knowledge. The following section discusses recent work related to SNNs.

2.5 Simulator Neural Networks

SNNs are trained by using data, experimentally collected, from real-world experiments. SNNs have been utilized as simulators in the ER process to create controllers for tasks such as path following, obstacle avoidance, light approaching behaviour and inverted pendulum stabilisation [Pretorius *et al.*, 2013]. SNNs have been used to model the dynamics of miniature vehicles and rotorcraft [De Nardi, 2010]. Research has also shown that SNNs can simulate the dynamics of the pendulum swing-up problem [Nakamura, Saegusa, and Hashimoto, 2007].

The conventional approach to creating SNNs and evolving controllers using the ER process is as follows [Pretorius *et al.*, 2013]:

1. Randomly generated commands are performed on a real-world robot. As a result of these evaluations, the robot moves around the environment and data is collected by using motion tracking and sensor logging techniques.

2. When sufficient data has been collected, it is used to train SNNs to predict the real-world robot's motor and sensor behaviour.

3. The trained SNNs are used to evaluate the fitness of candidate controllers during controller evolution.

4. At the end of the evolutionary process, the best controller is transferred and evaluated on the real-world robot.

Advantages of SNNs are that they possess good generalisation abilities, prediction accuracy and can handle large quantities of noise [Pretorius *et al.*, 2013]. The construction of SNNs can be simpler than that of a physics-based simulator, because no prior knowledge is required of the physics governing the robotic systems. It has also been shown that SNNs can be computationally more efficient, more accurate and can result in greater transferability to reality when compared to physics-based models [Pretorius, du Plessis, and Gonsalves, 2014].

There are, however, certain disadvantages to developing simulators before the ER process. The simulator must be created before the ER process can be initiated. The simulator may be trained to simulate real-world behaviours that are often not required for the successful evaluation of controllers for the goal task. Many time-consuming real-world evaluations may be needed to create a simulator. These SNNs remain static and thus are unable to take into account changes due to mechanical wear or environmental factors. Once trained, the SNNs are only able to handle a specific robot and environment. Any changes in the environment or robot can result in the need to completely retrain the simulator. In light of all these disadvantages,

research into alternative approaches to SNN creation could be of benefit to ER. The following section discusses research related to bi-directional approaches for the development of simulators and controllers.

2.6 Concurrent Controller and Simulator Development

This section discusses bi-directional approaches to creating simulators and controllers. Evolving controllers, in reality, is not always feasible (Section 2.4). This leads to many ER experiments being conducted in simulation. Therefore, dealing with the reality-gap problem is an important area of research and many approaches can be taken. One obvious approach to eliminating the reality-gap problem is to evolve controllers on real-world hardware only [Floreano and Mondada, 1994]. A combined approach can also be taken where controllers are evolved in simulation then transferred to a real-world robot and the ER process is continued in reality [Pollack, Lipson, Ficici, Funes, Hornby, and Watson, 2000]. It may be more effective to evolve a simplified robot performing a simple task. Once an effective controller has been found, additional layers of complexity are added incrementally [Brooks, 1992].

Controllers are more likely to fail to cross the reality-gap if levels of noise in a simulator are not sufficient [Jakobi *et al.*, 1995]. This makes it necessary for the simulator to use a high level of noise or make use of data sampled directly from reality, however neither of these approaches scale well, as the complexity is increased [Bongard, 2013]. The inclusion of the correct amount of noise in simulation often requires many hardware evaluations and manual tuning of the simulator [Mouret *et al.*, 2012]. Approaches that require less experimentation and manual tuning would therefore be advantageous.

The one-directional transference of a controller from simulation to reality can

be replaced by a more bi-directional approach. Researchers have proposed bi-directional approaches that allow the optimisation process to alternate between the simulator and the real-world [Bongard, Zykov, and Lipson, 2006a; Cully, Clune, Tarapore, and Mouret, 2015; De Nardi and Holland, 2008; Grefenstette and Ramsey, 2014; Mouret *et al.*, 2012; Zagal and Ruiz-del Solar, 2007]. This allows for the collection of empirical data from real-world controller evaluations that continually improve the simulator. The improving simulator develops better controllers which, in turn, are evaluated in reality, providing more relevant feedback for simulator improvement.

It has been proposed that robots should be able to maintain internal models of themselves that can be used to predict a robot's own behaviour [Bongard, Zykov, and Lipson, 2006b]. Research has shown that using internal model based evolution can reduce the number of physical hardware evaluations needed to evaluate controllers [Keymeulen, Iwata, Kuniyoshi, and Higuchi, 1998]. A population of internal models can be maintained and evolved to better represent the real-world model, thus allowing a robot to maintain an internal model of reality that can be used to predict its own behaviour.

A physics model is a specific example of an internal model. The use of a physics-based simulator requires the tuning of many physics-model parameters. As robots and their environments become more complex, hand-tuning physics-based parameters becomes more difficult. To overcome this, research into bi-directional approaches for autonomously evolving the parameters of a physics-based simulator has shown much success in producing controllers that perform tasks adequately on real-world robots [Bongard and Lipson, 2004; Moeckel *et al.*, 2013]. A disadvantage of this approach is that a physics-based model still needs to be constructed from prior knowledge and parameters optimised based on real-world observations.

Bi-directional approaches minimise the number of real-world evaluations required to develop effective simulators [Bongard and Lipson, 2004]. There is also a reduction in the amount of manual intervention required to develop effective simulators

since much of the process is automated. Bi-directional approaches can automate the building of physics models. Inaccuracies can be identified by comparing robot behaviours in simulation and in reality. The observed inaccuracies are then eliminated by changing the simulator to match the real-world behaviours.

The following sections are all examples of bi-directional processes. The Anytime Learning Algorithm is discussed in Section 2.6.1. The Estimation-Exploration Algorithm (EEA) is described in Section 2.6.2. The Transferability Approach is described in Section 2.6.3. The Intelligent Trial-and-Error Learning approach maps the performance of various behaviours, which is discussed in Section 2.6.4. The Back to Reality Algorithm (BTR) is described in Section 2.6.5 and models based on empirical data are discussed in Section 2.6.6.

2.6.1 Anytime Learning Algorithm

The Anytime Learning approach has been applied to a cat-and-mouse case study [Grefenstette and Ramsey, 2014]. Tracker and target agents played the role of cat and mouse, respectively. The target agent follows a random speed and course while the tracker agent tries to keep within a certain distance of its target without getting too close. If the tracker is too close to the target, it will be detected and fail the task. Agents have a set of sensors, such as the bearing, heading, speed and range to the target agent. The tracker must control its direction and speed and is initially unaware of the target's detection range or speeds.

Grefenstette and Ramsey [2014] proposed the Anytime Learning approach for continuous learning in changing environments. The approach consists of a population of models and a population of controllers. The approach consists of two components (Figure 2.5): the execution system and the learning system. A controller consists of a knowledge base and decision maker. The knowledge base consists of sets of situation-response rules which affected the decision maker's strategies. The execution system makes use of a decision maker that controls robot behaviour in

the real-world environment, depending on the current strategy. Similarly, the learning system makes use of decision makers and knowledge bases to evaluate strategy performances in simulation.

The population of simulation model parameters is maintained and judged against real-world data to determine their accuracy and are updated when necessary by the monitor. The best model parameters are used by the simulator for controller evolution. The population of strategies contained in the test knowledge base is evolved in simulation by means of a Genetic Algorithm (GA). The learning system experiments with different strategies using the simulator to attempt to find a better strategy for controllers.

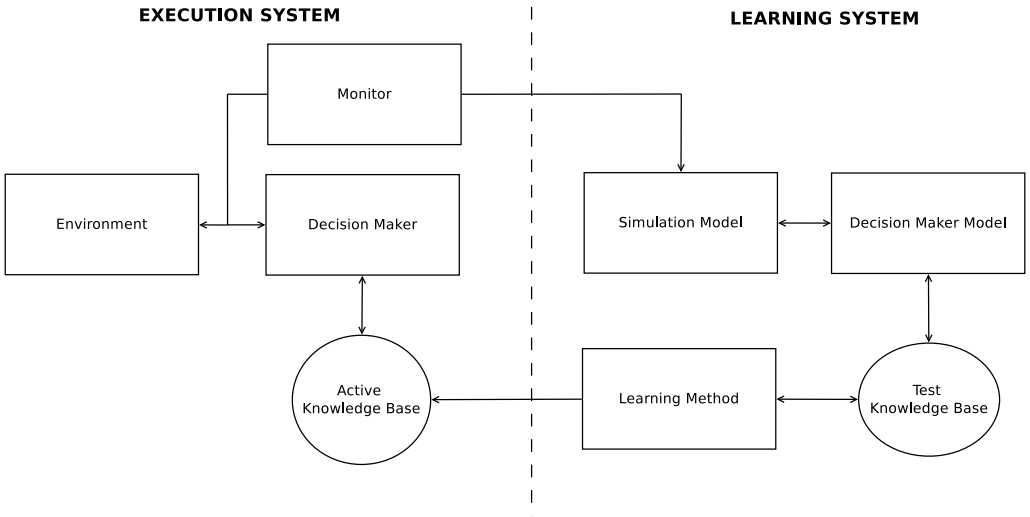


Figure 2.5: Anytime Learning System [Grefenstette and Ramsey, 2014]

Real-world robot evaluations are monitored and used to improve simulator models which, in turn, are used to learn improved strategies for the knowledge base. The monitor focused on measuring the real-world robot's speed which was used to estimate the mean and standard deviation parameters of the speed distribution used by the simulation model. Two types of updates occurred in the approach. Firstly, the learning system could find a better strategy for the decision maker (controller). Secondly, the monitor could update the parameters of the simulator model based on

the real-world observations. Once the simulator model parameters were updated, the learning system tried to find a better strategy for controllers.

The approach showed stable improvements of behaviours over time. Performance relied heavily on the design of the controller and the updating of the simulator based on real-world observations [Zagal and Ruiz-del Solar, 2007]. The researcher may require specialised knowledge about the dynamics of the robotic system being simulated to effectively update the simulator [Zagal and Ruiz-del Solar, 2007]. Anytime Learning requires the measurement of various real-world parameters that relate to the simulator but, these measurements could be difficult to relate to simulator parameters, such as friction, mass and many others.

2.6.2 Estimation-Exploration Algorithm

Bongard and Lipson [2005] proposed the Estimation-Exploration Algorithm (EEA) that integrates robotic self-modelling into the ER process (Figure 2.6). The EEA is a hybrid co-evolutionary algorithm for maintaining populations of models (estimates of what the robot morphology looks like) and populations of tests (controllers used to explore the simulator search space). After an adequate model is identified, controllers are evaluated by using this model for the purpose of accomplishing a particular goal task.

The EEA has been validated by using a four-legged, real-world robot with eight degrees of freedom (robot shown in Figure 2.6). The robot consisted of two tilt angle sensors and eight joint angle sensors. The goal task of the robot was to achieve forward locomotion.

The search space of possible models included a large number of arrangements of the limbs. The algorithms were shown to be effective in the handling of damage, such as the removal or shortening of limbs. If the robot was damaged, the damage was detected and models were improved in order to find a model that is better able to account for the damage.

The EEA uses a traditional physics-based simulator that models real-world phenomena. Models consist of a set of parameters for the various physical properties of the robot or its environment, such as leg positions, width and length of the body, weight distributions, inertia, gravity and so on.

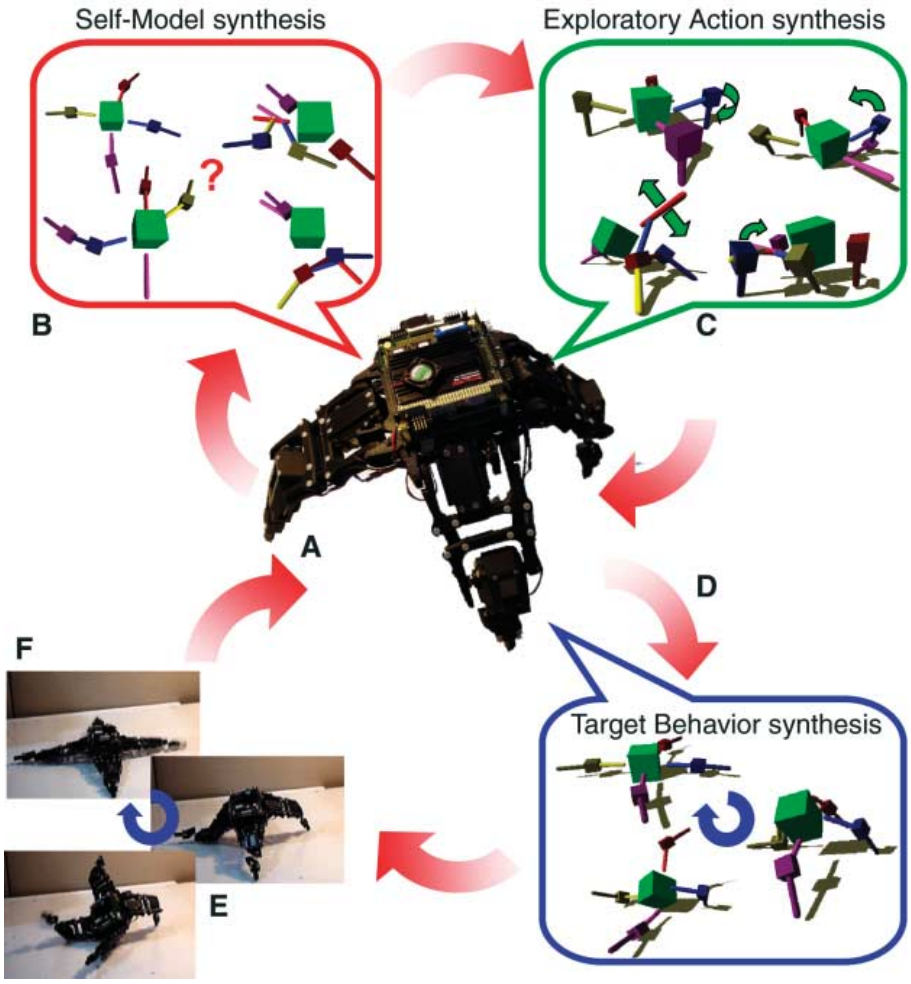


Figure 2.6: Estimation-Exploration Algorithm [Bongard *et al.*, 2006a]

The EEA has two phases, namely, the estimation phase and the exploration phase. For Figure 2.6, the estimation phase tests controllers designed to explore the search space on a real-world robot (**A**). The robot's real-world behaviours are compared to the simulated models and a population of models are optimised on their ability to predict real-world behaviours (**B**).

Next is the exploration phase. A population of test controllers are optimised based on their ability to create disagreement between models (**C**). This is to explore uncertainties in the population of models [Bongard and Lipson, 2005]. The test controllers seek to improve the simulator's ability to generalize and eliminate possible uncertainties in the modelling of reality. For example, a test controller is chosen because the controller's simulated behaviours are the most inconsistent amongst the population of simulator models. Once the test controller has been evaluated on the real-world robot, the models are improved to account for the observed behaviours. The exploration phase tries to find the next test controller that maximizes disagreement between models.

The algorithm cycles between the exploration and estimation phases until some stopping condition is met. Once a viable model is found, it is used to evolve controllers to achieve the desired behaviour (**D**) and the best controller in simulation is evaluated on the real-world robot (**E**). The cycle may continue to step (**B**) to improve models or to generate new behaviours (**F**).

A major advantage of the EEA approach is the ability to detect and adapt to unforeseen damage [Bongard *et al.*, 2006a]. The EEA was the first demonstration of a physical system being able to discern a sufficiently accurate model of itself without human intervention and with little prior knowledge [Bongard *et al.*, 2006a].

2.6.3 Transferability Approach

The Transferability Approach does not attempt to improve the simulator in any way, but rather attempts to complement it [Mouret *et al.*, 2012]. Simulators often have limitations in the modelling of certain phenomena in order to increase computational performance [Mouret *et al.*, 2012]. If the simulator's limitations are known during the ER process, controllers could avoid solutions that rely on dynamics that have not been accurately simulated.

The validation experiments employed a quadruped robot which had eight degrees

of freedom. The controller consisted of a sinusoidal equation that determined the angular positions of motors. The type of behaviours achieved by the robot were dependent on the parameter settings of the sinusoidal equation. The robot's goal task was to cover the largest possible distance in 10 seconds. The experiments initially followed a typical ER approach and controllers were evolved in simulation and the final solution was evaluated on a real-world robot. The observed results of this traditional approach showed a clear reality-gap problem. The follow-up validation experiments made use of the Transferability Approach and also compared it to various other approaches, such as the traditional ER approach, direct evolution on the robot and surrogate modelling of the fitness function. The results of the experiments demonstrated that the Transferability Approach generally obtained the fittest solutions in reality and resulted in the closest similarity between simulation and reality.

The Transferability Approach proposes a multi-objective fitness function composed of two parts. One part estimates how well a solution may transfer from simulation to reality (called the transferability function) and another part estimates how well the desired behaviour is achieved in simulation [Koos, Mouret, and Doncieux, 2013]. The transferability of solutions can be measured in various ways, such as by comparing the trajectory of the centre of mass, angular positions of joints, contact times of legs with the ground or the distances covered in simulation and reality. The inputs for the transferability function can be the genotype of the controller. Alternatively, the transferability function could take as input the behaviours observed in simulation that do not match up to reality, such as the robot jumping unrealistically when limbs hit the ground. The transferability function for the experimental work took as input the 3D-trajectory behaviour of the centre of mass of the robot in simulation (calculated as the summed Euclidean distance between each point's position in reality and simulation along the robot's trajectory).

It may be difficult to map the controller genotype space to the transferability measurement because the relationship may be highly complex. Many genotypes

would result in a large number of inputs, such as ANN controllers. Alternatively, deciding on which particular behaviour to map for the transferability measure could require knowledge of the weaknesses of the simulator. Effective transferability measurements are highly dependent on the chosen task and on the simulator.

A physics-based simulation was used to estimate the distance covered by the quadruped robot when evaluated. The transferability function was used to estimate how well a controller's behaviour transferred from simulation to reality. The transferability function can be implemented by means of an ANN, Support Vector Machine or any other regression/interpolation method which can be trained to predict the differences between the robot's behaviour in simulation and reality through some transferability measure [Mouret *et al.*, 2012]. The experimental work used the Inverse Distance Weighting method to model the transferability function [Mouret *et al.*, 2012]. The transferability function can be trained either continually or before the ER process has begun. The continual training approach is the preferred method because the controller candidates, used to generate training data, are closer to the desired behaviour [Mouret *et al.*, 2012].

2.6.4 Intelligent Trial-and-Error Learning

Cully, Clune, Tarapore, and Mouret [2015] have demonstrated an Intelligent Trial-and-Error Learning algorithm that utilizes a pre-computed behaviour performance map of a large number of behaviours. The behaviour performance map stores how well the robot performs for a given behaviour. This approach is mainly designed to allow a robot to compensate for unanticipated damage without a lengthy self-diagnosis procedure or the need to update the simulation model. The Intelligent Trial-and-Error approach was also successful in allowing the robot to adapt to changing environments.

The approach was validated by using a hexapod robot with 18 degrees of freedom. The controller consisted of a gait with 36 parameters that described the movement of

joints. Behavioural measures was a six-dimensional space that defined the duration of contact each robot leg had with the surface for the particular controller and the corresponding performance was the speed the robot. An optimization algorithm was used to search for a large number of high performance controllers using a standard physics-based simulator and a behavioural performance map was generated that assigns fitnesses to the various behavioural measures.

When the behavioural performance map has been computed, the map can be used to search for high performance solutions to be evaluated on a real-world robot. Initially, the predicted performances stored in the map have a low confidence level as the real-world performance is uncertain. Once a controller has been evaluated in reality, the map can be updated with the real-world performance and given a high confidence. During updates, similar nearby behaviours are also adjusted with confidence values proportional to how close the behaviours are to the behaviour evaluated in reality. For example, if the performance of a given behavioural measurement was obtained by evaluating a controller on the real-world robot, then the behavioural performance map was updated by using the real-world performance and similar behaviours were also updated based on how close they were to the evaluated behaviour.

Once the behavioural performance map had been computed, the robot was shown to quickly develop effective controllers in less than 30 seconds for the undamaged robot and just over a minute for the damaged robot. A disadvantage of this approach is that it requires the evaluation of a large number of behaviours in simulation before the behavioural performance map is generated and controllers can be evaluated in reality. However, the performance map only needs to be generated once and it can be saved for future use.

2.6.5 Back to Reality Algorithm

Zagal and Ruiz-del Solar [2007] proposed the Back to Reality (BTR) algorithm. The

BTR algorithm co-evolves populations of controllers and simulators. The differences between controllers evaluated in simulation and in reality are identified and used as feedback for simulator improvement. A physics-based simulator and the Sony AIBO four-legged robot were utilized for the validation experiments. The simulator consisted of 12 parameters which defined the mass distributions, weights, frictional properties, gravity and the model of the joint torque and model parameters. The simulator optimization process tries to improve the parameters of the physics model. Controllers consisted of a set of 20 or 25 parameters (depending on the given goal task) that defined the robot's gait. Controllers were optimised either for walking speed or for kicking a ball.

The fitness function used to evolve simulators is the average difference between a controller's fitness in simulation versus reality [Zagal and Ruiz-del Solar, 2007]. The simulator optimization process tries to minimize $\triangle fitness$:

$$\triangle fitness = \frac{1}{m} \sum_{k=1}^{m} |f_{sk} - f_{rk}| \tag{2.10}$$

A GA selects the best m controllers and each is evaluated on a population of simulators. For each simulator, the m robot controllers are evaluated and given corresponding fitness values f_{sk} (s: simulator; k=1,...,m). The controllers are also evaluated in reality and have fitness values f_{rk} (r: reality; k=1,...,m). A GA is also used to evolve a population of simulators by using equation (2.10) as the fitness function.

BTR has been shown to require fewer real-world evaluations than many other bi-directional approaches [Zagal and Ruiz-del Solar, 2007]. The BTR approach is not reliant on any particular simulation approach. The BTR algorithm is simple and requires only the $\triangle fitness$ as a fitness function for optimizing the population of simulators. This is in contrast with many existing methodologies which require the explicit measurement and comparison of simulator related variables [Zagal and Ruiz-del Solar, 2007].

2.6.6 Model-fitting based on Empirical Data

Kamio and Iba [2004] and De Nardi and Holland [2008] propose that models can be automatically constructed with little prior knowledge by collecting experimental data during real-world controller evaluations. The experimental data is used to autonomously build models that simulate robot behaviours. These models are used to evolve future controllers for achieving specific behaviours.

$$s_{t+1} = f(s_t, a_t) \tag{2.11}$$

The real-world environment can be modelled by means of prediction models (Equation (2.11), [Kamio and Iba, 2004]), which are expressed as a state of the environment (s_t), an action (a_t) and the resultant state after an action is applied (s_{t+1}). Prediction models are created by using reinforcement learning techniques [Kamio and Iba, 2004] or evolutionary algorithms [De Nardi and Holland, 2008] and data obtained from real-world controller evaluations.

De Nardi and Holland [2008] synthesized a set of non-linear differential equations that represented the dynamic models of a miniature rotorcraft. The rotorcraft robot consisted of four motors that were fitted with propellers. The linear and angular accelerations for the body coordinates of the robot were derived by using Genetic Programming techniques. This allowed for both the parameters and structure of the acceleration models to be developed autonomously by using real-world data.

Kamio and Iba [2004] used GP or Cluster Approximation methods for developing simulators that were able to predict behaviours of a humanoid robot whose goal task was to move a box to a specific goal marker. Actions allowed by the humanoid robot were basic movements such as sidestep left/right, move forwards six steps, turn right/left and a few combinations of turning and sidestepping. The simulator predicted the positions of the box and goal marker.

Two main processes are interleaved with each other. Models are evolved to predict real-world behaviours and test controllers are evolved for a particular purpose.

The prediction models which represent the simulator can be implemented by using Genetic Programming, Clustering Approximation or other model-fitting techniques [De Nardi and Holland, 2008; Kamio and Iba, 2004].

Data is collected during real-world controller evaluations and can be continually included in the simulator training process. Since training data containing real-world noise is continually added, over-fitting is avoided [Kamio and Iba, 2004]. These approaches require few physical assumptions and eliminate biases in the development of simulators.

2.7 Conclusions

Evolving controllers purely on real-word robots is often not a feasible option. Therefore, simulators are used to accelerate the ER process, but this results in much time being devoted to the design, creation and manual tuning of simulators that allow controllers to successfully cross the reality-gap. Much of the research on the bi-directional approaches for improving simulators and controllers is focused on reducing the reality-gap problem, automatic damage recovery and automatic parameter tuning of models. Little research has been conducted on the autonomous creation of simulators with no prior knowledge during the ER process. Existing approaches for the concurrent creation of controllers and simulators still require manual intervention for the creation of the initial simulators, with the exception of empirical methods. Specialised knowledge is usually required for the manual creation and improvement of these simulators.

This research seeks to satisfy the need for the autonomous creation of simulators by utilizing experimentally collected data with minimal human intervention or specialised knowledge. Traditional SNNs are simple to construct, are arguably easier to automate than physics-based approaches and require no prior knowledge about the dynamics of the robotic system. The bi-directional development of non-SNN based simulators and controllers has been shown to be an important area of re-

search in ER. However, no research has been conducted on evaluating the feasibility of the concurrent creation of SNNs and controllers in the ER process. The following chapter proposes an approach that combines SNNs with a bi-directional approach.

Chapter 3

PROPOSED APPROACH AND EXPERIMENTAL METHOD

3.1 Introduction

Pre-computed SNNs have shown much potential as robotic simulators due to their simple construction and minimal requirements of expert knowledge (Section 2.5). Bi-directional approaches have shown much promise in automating and speeding up the development of simulators (Section 2.6). The developed approach is discussed in Section 3.2 and considerations that motivated the approach developed in this research is described in Section 3.3. The experimental method used to validate the viability of the proposed approach is addressed (Section 3.4). Lastly, conclusions are drawn in Section 3.5.

3.2 Proposed Approach

The contribution of this research is the development of a novel approach to SNN and controller development. The conventional approach to creating SNNs and evolving controllers during the ER process requires a sequential two-stage process. The first stage is the data collection process which requires the evaluation of many randomized

controllers on the real-world robot and the collection of the observed behaviours. Once enough data has been acquired, the SNNs are trained until they are sufficiently accurate. The second stage uses the trained SNNs to simulate the robot's behaviours during the normal ER process. The approach developed in this research combines the normal ER process and the data collection and training process (Figure 3.1).

The proposed approach developed in this research work will be referred to as the Bootstrapped Neuro-Simulation (BNS) approach. The BNS approach involves a bootstrapping process whereby controllers are continually evaluated on a real-world robot, thereby generating training data for the improvement of the SNNs. As the SNNs are improved, so too are the controllers. The improved controllers can then be evaluated on the robot to provide further training data to the SNNs and the cycle continues.

The BNS approach is illustrated in Figure 3.1 and consists of the following steps:

1. Controllers are selected for real-world evaluations from the population of controllers. Behavioural data from real-world evaluations is collected and stored in a training data buffer.

2. The training data stored in the training data buffer periodically integrates into SNN training to better predict robot behaviours.

3. The training SNNs are periodically copied and used to replace the controller evolution SNNs. The population of controllers is evolved using the controller evolution SNNs to estimate controller fitness.

4. The process continually repeats by going back to step 1 or is terminated when stopping conditions are met.

The BNS approach improves the SNNs and controllers concurrently so that controllers, evaluated in reality, would ideally converge towards the target behaviours. Consequently the SNNs are specialised to accurately model the target behaviours and are less focused on modelling unnecessary behaviours. BNS can speed up the

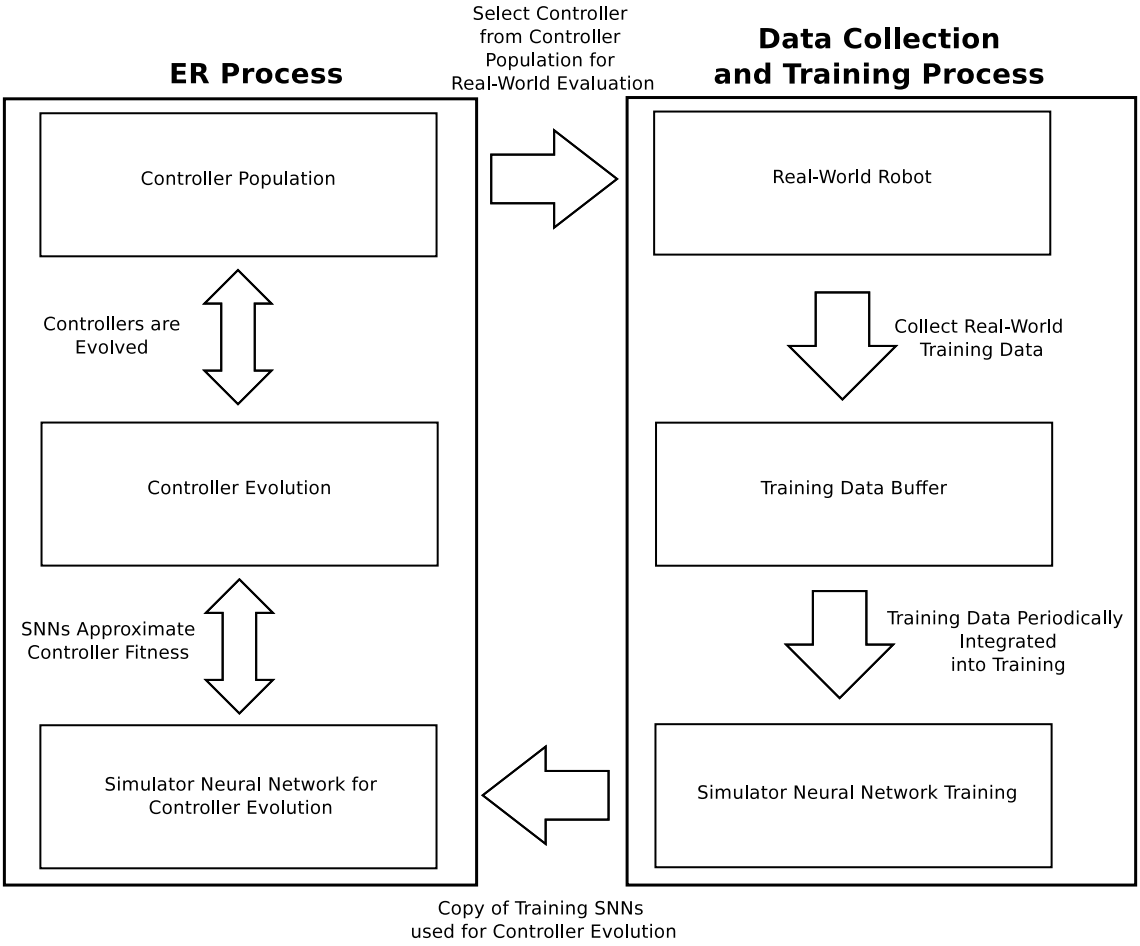


Figure 3.1: Bootstrapped Neuro-Simulation

ER process by eliminating the need to pre-compute SNNs before controllers are evolved and may require fewer real-world evaluations than pre-computed SNNs.

If the robot morphology or environment changes in any significant way, there would be no need to retain an existing simulator since the BNS approach automatically creates a new simulator and accounts for any changes to the system. The BNS approach could also be modified to potentially possess other useful properties such as the ability to adapt to changing environments or automatic damage recovery.

The proposed approach does have a few disadvantages that will now be discussed. The simulator developed with the BNS approach would be trained with specialised behavioural data which could result in a lack of simulator generality compared to

pre-computed SNNs. The developed SNNs could accurately predict the behaviours required for the task chosen during data collection but could be unable to accurately predict the behaviours of an unseen task. SNN approaches may not be a viable option for certain robots, environments or tasks. The BNS approach could also converge on a sub-optimal solution and be unable to find an appropriate solution.

3.3 Motivations

It is hypothesised that a simulator can be developed during the ER process and little prior knowledge about the dynamics of the robotic system need be known. A key consideration is that the ER researcher would not be required to have extensive specialised knowledge of the dynamics involved in simulating the robotic system. Therefore, the use of standard physics-based modelling approaches that make use of prior knowledge is avoided. Hybrid models that combine empirical and physics-based approaches will also be avoided.

The main concern in this research is the creation of task specific controllers, so this study is less interested in the physical interpretations of underlying systems. Therefore, simulation approaches that model all the various components of a system using prior knowledge and structures that have real-world interpretations will be deemed unnecessary. These knowledge-based representations can be replaced with ones that are more behavioural-based in order to focus on the goal of developing specific behaviours for robots. As previously mentioned (Section 2.4), behavioural-based representations focus on modelling the behaviours of the underlying system and are easier to construct since the models are based on real-world observations. Therefore, a black-box modelling of robot behaviours should be used, such as SNNs. The chosen simulation approach in this work uses SNNs due to the success achieved in previous work.

The training of SNNs requires the sampling of real-world experimental data which is accomplished by performing many randomized movements on a real-world

robot. This sampling process is time-consuming and may damage the robot due to wear. The simulator is also trained to simulate robot behaviours not required for the development of the target robot behaviours. The simulator also needs to be created before the ER process can begin.

The bi-directional approaches discussed in Section 2.6 mostly require the development of an initial physics-based simulator. These bi-directional approaches can be broadly classified according to the level of involvement in improving the effectiveness of the simulator. Complementary approaches do not actually improve an existing simulator but merely assist in its effectiveness, such as the Transferability (Section 2.6.3) or Intelligent Trial-and-Error Learning (Section 2.6.4) approaches. Optimisation approaches such as Anytime Learning (Section 2.6.1), EEA (Section 2.6.2) and Back to Reality (Section 2.6.5) are actively involved in the improvement of simulator models. Lastly, a ground-up approach can be taken where the simulator is created and improved without the use of prior knowledge (Section 2.6.6). The BNS approach (Section 3.2) developed as a core part of this research falls into the ground-up classification. The following section discusses the experimental method used for demonstrating the viability of the BNS approach.

3.4 Experimental Method

This section discusses the experimental method used for demonstrating the viability of the BNS approach. The experimental method used to identify influential factors related to its success or failure is also discussed.

The viability of the BNS approach was validated by using a differentially-steered mobile robot and a snake-like robot (Chapters 4 and 5, respectively). Each robot performed different trajectory planning tasks of varying levels of complexity. ER was used to evolve navigational controllers (Sections 4.2.2 and 5.3.2) in simulation while, simultaneously, the simulator was optimised by using resilient backpropagation training (Sections 4.2.3 and 5.3.3). Controllers were periodically selected from

the population of controllers and evaluated on a real-world robot. These evaluations were tracked by using motion tracking techniques and the behavioural data generated was collected and used for training the simulator. The simulator used in the controller evolution process would thus commence as untrained and would gradually improve as more behavioural data was collected.

The performance of the BNS approach for various parameter settings was subsequently investigated. This experimental investigation required the testing of a large number of parameter combinations which was made possible by using an already trained simulator as a substitute for the real-world robot. A pre-computed simulator from previous studies was used for the first prototype (Chapter 4), whereas a pre-computed simulator was not available for the second prototype (Chapter 5) and was developed as part of this research (Section 5.4). The substitute (simulated) real-world robot expedited the process and made the parameter comparison experiments viable. It was anticipated that the diversity of the controller population was a key factor because it governed the controller search space explored during the ER process and also influenced simulator training. The analysis was thus focused on how the parameters affected diversity and how diversity, in turn, manifested into the success of the controllers. The diversity in relation to the effectiveness of the simulator to predict robot behaviour was also analysed. The investigation of parameter settings was successful in determining influential factors related to the success or failure of the BNS approach. Sections 4.5 and 5.8 describe how parameters were investigated while Sections 4.6 and 5.9 discuss the results.

3.5 Conclusions

The use of bi-directional approaches for improving simulators and controllers has been shown to effectively resolve the reality-gap problem and help automate the simulator development process. Many of the existing bi-directional approaches still require much human intervention in the creation and design process of simulators.

Specialised knowledge is usually required for the creation and improvement of these simulators.

This research aims to automate the creation of simulators by utilising experimentally collected data with minimal human intervention or specialised knowledge. SNNs have shown much promise in being able to achieve these goals. The concurrent creation of non-SNN-based simulators and controllers has also shown promise (Section 2.6). However, no known research has been conducted on evaluating the feasibility of the concurrent development of SNNs and controllers in the ER process which is proposed in this work. The BNS approach was developed to investigate the viability of concurrently developing SNNs and controllers in the ER process. The following two chapters are aimed at validating the feasibility of the BNS approach by using two types of robot morphologies and determining the relevant influential factors.

Chapter 4

KHEPERA PROTOTYPE

4.1 Introduction

This chapter covers the experimental work performed on a differentially-steered mobile robot. The procedures used for investigating the proposed BNS approach are discussed in Section 4.2 and details of the prototype experiments are discussed in Section 4.3. The results of the experimental work are presented in Section 4.4. The parameter comparison experiments are discussed in Section 4.5 and the results are given in Section 4.6. Lastly, a discussion of the work in this chapter is given in Section 4.7.

4.2 Experimental Procedure

This section covers the experimental work conducted on the Khepera robot. Section 4.2.1 discusses the hardware and data capturing techniques. Details of the controller are discussed in Section 4.2.2 and the developed simulator is addressed in Section 4.2.3.

4.2.1 Hardware and Data Capture

The robot used for this experimental work is the Khepera III mobile robot (Figure 4.1). The reason why this robot was chosen is because of the frequent use of the robot in ER [Floreano and Mondada, 1994; Floreano *et al.*, 2008; Koos *et al.*, 2013; Miglino *et al.*, 1995] and ease of use. The robot is differentially-steered and movement is controlled by two separately driven wheels. The robot is controlled by sending commands over a Bluetooth serial interface. Steering is accomplished by varying the relative rate of rotation of the wheels. The environment upon which the robot operated was a horizontal skid-proof surface. The Khepera was modified by mounting three LED infra-red lights on top. A Nintendo Wii [Nintendo, 2014] remote was positioned approximately 1.8 meters above the robot and software was used to track the LEDs so that behavioural data could be captured [Lee, 2008].

Figure 4.1: Khepera III mobile robot [K-Team, 2014]

4.2.2 Controllers

The controllers developed for the Khepera opted for open-loop navigation. A controller in this research is defined as a list of commands that, when executed on the

robot, would cause it to manoeuvre along a particular trajectory without feedback. Open-loop navigation was chosen due to its simplicity. Controllers, therefore, did not have access to the robot's real-time position during evaluation. Controllers consisted of a variable length, sequential list of commands, each consisting of the left motor speed, right motor speed and the time duration for which the command must be executed (Figure 4.2). This was a first-time study into the viability of the BNS approach and in order to keep the investigation simple, motor speeds were restricted to moving forwards only. The BNS approach began by creating a population of randomly generated controllers. Between 4 and 13 commands were initially generated for each controller in the initialised population, but subsequent generations could create controllers that could contain an unlimited number of commands through cross-overs. A controller was evaluated by executing the list of commands sequentially on the robot. Collisions with the boundaries of the working surface were not considered.

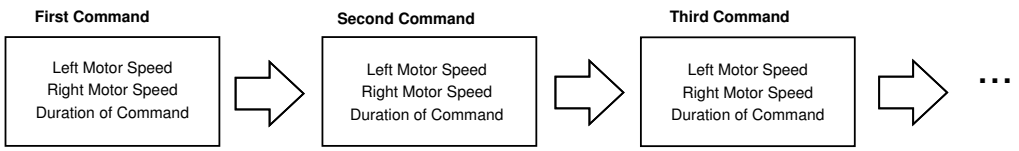


Figure 4.2: Khepera controller morphology

Controllers were developed for three different trajectory planning tasks. Figures 4.4 to 4.6 demonstrate the paths followed by the robot for each task. The robot's starting position was located centrally amongst the goal points and faced northward. Each of the three tasks was described as a sequential list of goal points on the operating surface. Success was accomplished when the robot traversed all these points in the given order. The robot was judged to have reached a given goal point if it moved within seven centimetres of the point. The seven centimetre leniency was chosen to allow for more leeway in reaching goal points during controller evolution. The first task, referred to as Task 1, was a circular path (Figure 4.4). The second task, referred to as Task 2, was an infinity sign path (Figure 4.5). The third task,

referred to as Task 3, was a complex path (Figure 4.6).

The end result of a controller evaluation was a list of positions reached by the robot at the end of each command. There was also a list of goal points the robot needed to reach in the specified order. The pseudo-code for how controller fitness was calculated is shown in Algorithm 2. This algorithm took as input the list of positions and goal points. During the traversal of the positions' list, the distance to the first goal point was calculated. Once that goal point was reached, the distance to the next goal point in the list was used and so on. For each of the positions in the list, the distance to its goal point was calculated. These distances were summed together and a penalty distance was added for each missed goal point. This total distance would ideally be minimized during controller evolution and the fitness was calculated as its inverse.

Algorithm 2: Controller fitness evaluation

Data:

positions ← List of positions reached by robot

goalpoints ← List of goal points

Result: Fitness of controller

penalty ← A large constant integer value

$goalPointIndex \leftarrow 0$

$sum \leftarrow 0$

for *each position in positions* **do**

  if *position reaches* $goalpoints[goalPointIndex]$ **then**

    $goalPointIndex \leftarrow goalPointIndex + 1$

  $sum \leftarrow sum +$ (distance to $goalpoints[goalPointIndex]$)

$sum \leftarrow sum + penalty$ * (number of goal points not reached)

$fitness \leftarrow 1/sum$

return *fitness*

The fitness function was designed to minimize the number of commands used to accomplish the task due to summing of the distances to the next goal point

previously mentioned, and fewer commands result in fewer summations. Controllers were continually evolved throughout the process using a GA. Elitism was used, and the best performing controller for each generation continued over to the next generation.

4.2.3 The Simulator

The operating surface can be mathematically represented as an xy-plane of a Cartesian coordinate system, referred to here as the global coordinate system. The robot's movements were captured based on the global coordinate system. The robot had its own local coordinate system where the origin was always the centre of the robot's wheel axis and the y-direction was aligned with the robot's forward heading. The simulator was trained to predict changes in the robot's position according to the local coordinate system.

The simulator was used to assign fitness values to the population of controllers. In the beginning, the simulator was untrained, so fitness assignments would effectively be random. However, the quality of the fitness assignments gradually improved as more training data became available from real-world controller evaluations. Training patterns consisted of the previous left motor speed, previous right motor speed, current left motor speed, current right motor speed, time duration of execution, the robot's local x-displacement, the robot's local y-displacement and the change in the robot's orientation angle, which was added to the training data set or verification data set of the simulator. There was a 75% probability of data being added to the training data set and a 25% probability of data being added to the validation set. The validation set was not utilized during simulator training, but was used to analyse potential issues in over-fitting and accuracy. It was found that over-fitting did not occur during training.

The simulator consisted of three separate SNNs, each consisting of a single Feed-Forward Neural Network (FFNN) (Figure 4.3). The FFNN setup was chosen based

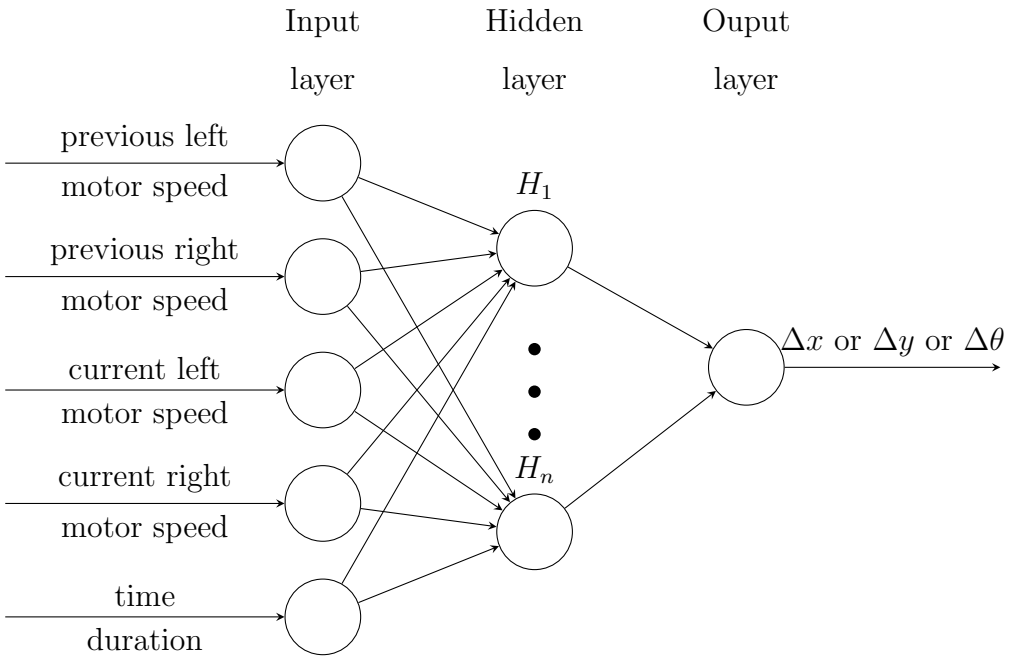


Figure 4.3: Simulator Neural Networks of the Khepera robot

on previous investigations. The simulator was separated across multiple FFNNs to produce more accurate results [Pretorius, du Plessis, and Cilliers, 2009]. Each FFNN had 5 input neurons, 20 hidden neurons and 1 output neuron. An indication of the optimal number of hidden neurons was determined experimentally during previous studies [Pretorius, 2010]. The input and output neurons made use of a linear activation function while the hidden neurons made use of a sigmoid activation function.

All FFNNs took as input the previous left motor speed, previous right motor speed, current left motor speed, current right motor speed and the time duration of the current command. The outputs of each of the separate FFNNs were the x-displacement (Δx), y-displacement (Δy) and change in angle ($\Delta\theta$).

SNNs were implemented using an open source machine learning library called Encog [Heaton Research, 2014] and were trained using the Resilient Backpropagation algorithm [Riedmiller and Braun, 1993]. There were two types of SNNs, the training SNNs and the controller evolution SNNs. A copy of the training SNNs

was periodically made and used for the controller fitness assessments (referred to as the controller evolution SNNs). The training SNNs were periodically trained for 1000 iterations by using Resilient Backpropagation, after which new training data was added (if available) from the training data buffer and training iterations were continued. Noise was not added to the controller evolution SNNs for any of the experiments presented in this investigation, because it was observed that controllers were able to transfer well to reality without the addition of noise.

4.3 Prototype Experiments

The BNS approach was validated through an experimental prototype that made use of the real-world hardware mentioned in Section 4.2.1. Experiments proceeded until the robot was visually perceived to have consistently traversed all the goal points accurately. Each of the three trajectory planning tasks was completed by using the BNS approach over three trials.

The parameters used for controller evolution are given in Table 4.1. Optimal parameter values were determined experimentally based on the parameter comparison experiments described in Section 4.5.

Controller Population Size	400
Initialization	Random from a uniform distribution
Selection	Tournament (Tournament size 50%)
Cross-over Method	Two parent
Mutation Rate	10%
Mutation Method	Random Component Perturbation

Table 4.1: Parameters for controller evolution

The controller population size parameter (Table 4.1) was the size of the population of controllers used for controller evolution. The initial population of controllers was created such that each controller contained a random number of between 4 and 13 commands, each command being generated randomly from a uniform distribution. The range of motor speeds was based on the operational limits of the robot and was between 3000 and 30000 units (scale used by the software) and the duration of each command was between 400 and 3000 milliseconds. Durations below the lower limit would be difficult to measure and predict accurately while durations above the upper limit could result in rotation angles that were larger than 180 degrees.

The selection operator used for controller cross-over was tournament selection [Engelbrecht, 2007]. A tournament selection size of 50% of the controller population was chosen and a two-parent cross-over method was used for controller evolution. Two-parent cross-over involved two parent controllers, for which the command sequence sizes of both parents may be of differing lengths. A random cross-over point was chosen for each parent which resulted in each parent forming two segments of commands. The child controllers were created by interchanging the first segments of each parent. There was an 80% probability that the cross-over points of two parents were identical and a 20% probability that they could be different. This cross-over method was chosen to allow for the number of commands to be optimised while not unnecessarily distorting existing good solutions. The mutation rate was the probability that the left motor speed, right motor speed or duration of a command might change by a random amount. Ranges for the mutations were between -2000 and 2000 for the motor speeds and the time durations were changed by -400 or 400 milliseconds. The mutation rate chosen for controller evolution was 10%.

An additional parameter which was specific to the BNS approach was the real-world selection parameter. At the end of every real-world controller evaluation a new controller was selected for the next real-world evaluation. This new controller was selected from the population of controllers by using tournament selection and the tournament size was specified by the real-world selection parameter. A real-world

selection size of 70% of the controller population was chosen for the experimental work that is presented in the following section.

4.4 Prototype Results

The real-world prototype experiments were able to demonstrate that the BNS approach is indeed viable. The developed SNNs made sufficiently accurate predictions of robot behaviour. This allowed controllers to successfully evolve and demonstrate the required behaviours.

The three trajectory planning tasks are presented in Figures 4.4 to 4.6. These figures represent the final controllers found for each trial run using the BNS approach. Real-world evaluations represent the number of controllers that were evaluated on the real-world robot. Each experiment was run until the robot was visually perceived to have successfully and consistently performed the goal task, which was why the total number of real-world evaluations for the trials differ. In each figure, the solid line represents the real-world path followed by the robot when the fittest controller was evaluated. The dotted line represents the developed simulator's perception of how the robot would behave when the same controller was evaluated in reality. The circled grey regions are the goal points with a radius of 7cm. The order in which the goal regions must be traversed by the robot are annotated with circled numbers. The total run-time and number of real-world evaluations for each trial is given in Table 4.2. It can easily be seen that the prototype run-times do not seem to be consistent and do vary considerably which may indicate that the BNS approach has room for improvement.

As previously mentioned, controllers were developed for open-loop navigation. Controllers were not aware of the position of the robot during evaluations which could result in an accumulation of errors. The robot's starting position was located centrally amongst the goal regions facing northwards. The prototype experiments showed that the trained SNNs were able to predict robot behaviour with enough

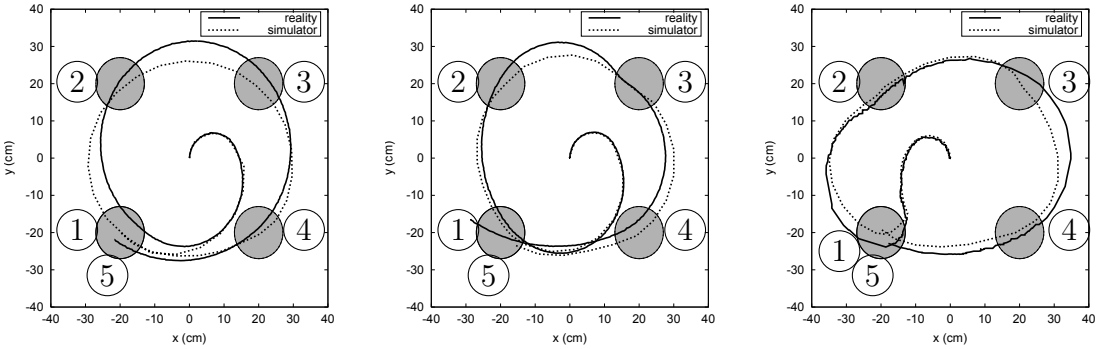
Goal task	**Trial number**	**Experiment run-time**	**Real-world evaluations**
Task 1	Trial 1	25 minutes	45
	Trial 2	12 minutes	20
	Trial 3	9 minutes	15
Task 2	Trial 1	25 minutes	40
	Trial 2	19 minutes	30
	Trial 3	12 minutes	20
Task 3	Trial 1	40 minutes	55
	Trial 2	50 minutes	60
	Trial 3	30 minutes	45

Table 4.2: Prototype run-times

accuracy to develop viable controllers.

Task 1 (Figure 4.4) had four goal points positioned 40cm apart from each other in a square formation. The first and second trial runs for Task 1 (Figures 4.4a and 4.4b) had similar solutions, where the robot made an initial right turn and produced a spiral shape. The third trial made use of an alternative strategy which differed from the first two trials by initially turning left instead.

Task 2 (Figure 4.5) had five uniquely positioned goal points where the centre point had to be reached twice by the robot. All solutions to the second task made use of the same strategy. The robot started at the centre position and traversed the goal regions in the annotated order. The real-world paths for the first two trials closely matched the simulator's perception of reality (Figures 4.5a and 4.5b). The third trial run successfully performed the task in reality, however, the simulator's perception of reality was less accurate (Figure 4.5c).

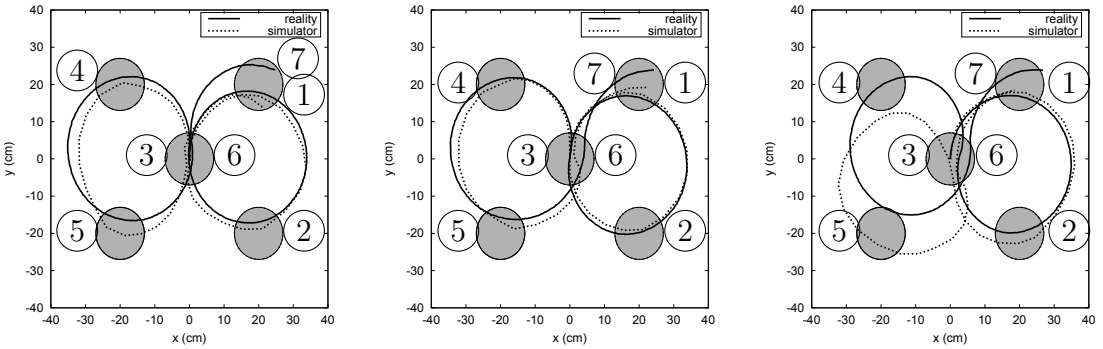


(a) First trial run at the 45th real-world evaluation

(b) Second trial run at the 20th real-world evaluation

(c) Third trial run at the 15th real-world evaluation

Figure 4.4: Task 1 prototype trial runs



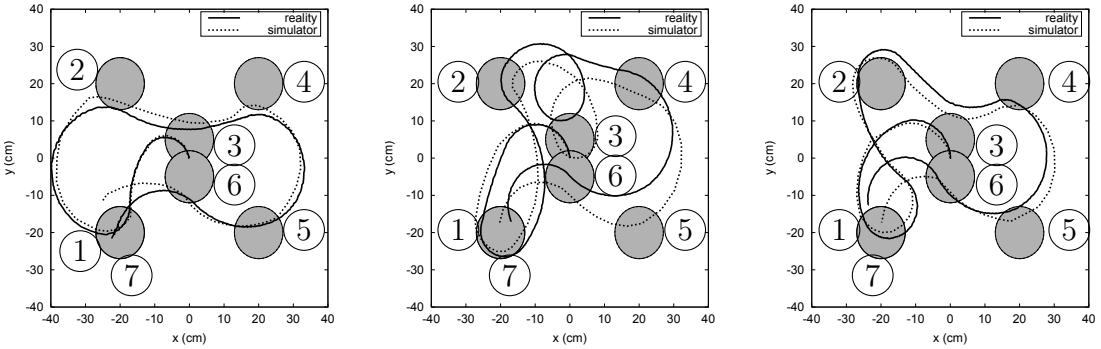
(a) First trial run at the 40th real-world evaluation

(b) Second trial run at the 30th real-world evaluation

(c) Third trial run at the 20th real-world evaluation

Figure 4.5: Task 2 prototype trial runs

Task 3 (Figure 4.6) contained six independent goal points and good solutions were needed to perform many sharp and quick turns. Each trial solved Task 3 by using a different strategy. The first trial contained no loops while the second and third trials contained two and one loops, respectively. The second and third trials contained a loop around the bottom left goal point. The second trial produced a loop around the top centre goal point. The developed simulator accurately predicted the robot's behaviours. The first and third trials were the most accurate and the second trial was the least accurate.



(a) First trial run at the 55th real-world evaluation

(b) Second trial run at the 60th real-world evaluation

(c) Third trial run at the 45th real-world evaluation

Figure 4.6: Task 3 prototype trial runs

During the BNS approach, controllers evaluated in reality would converge towards the desired behaviour. However, there were instances where this convergence was lost, but gradually improved towards the desired behaviour again. This sudden loss of successful controllers was possibly because the simulator periodically changed. Controllers that barely reached certain goal points could fail to reach them when the simulator was updated.

4.5 Parameter Comparisons

In order to investigate important factors that contribute to the success or failure of the BNS approach, the influence that various parameter settings had on success was investigated. The parameters tested are listed in Table 4.3. Due to the stochastic nature of the experiments, every possible combination of the parameter settings was independently run for 30 trials for each task.

The investigation of such a large number of parameter combinations on a real-world robot would take infeasibly long due to the time required to run them. A pre-computed simulator based on previous studies, referred to as the static simulator, was thus used as a substitute for the real-world robot [Pretorius, 2010]. Training

Controller Population Sizes	50, 100, 200, 400
Tournament Selection Sizes	10%, 20%, 30%, 40%, 50%
Mutation Probability Rates	10%, 30%, 50%, 70%, 90%
Real-world Selection Sizes	10%, 30%, 50%, 70%, 90%

Table 4.3: Parameter values used for comparisons

data generated from the real-world robot would inevitably contain errors due to inconsistencies in motor function and sensor readings [Pretorius, 2010]. To accurately simulate reality, noise was thus added to the static simulator in order to realistically replicate noise that would be present. The noise distribution was assumed to be Gaussian with mean zero because the real-world noise data had a mean close to zero and with variances based on the difference between the static simulator and real-world data. The static simulator with noise which acted as a substitute for the real-world robot will be referred to simply as the substitute real-world.

The substitute real-world also simulated the actual time required to evaluate controllers in reality. This artificial slowing down of the substitute real-world was done to allow for the SNN training and controller evolution to progress with durations similar to reality. Each experimental run continued until the hundredth controller was evaluated in the substitute real-world. Achieving the desired behaviour beyond the hundredth evaluation was considered impractical for real-world experiments.

A controller successfully completed a given task if during evaluation the robot passed through all goal points in the assigned order. Goal points were defined as points in the xy-plane forming a circled region with a radius of seven centimetres. Success was defined as a trial run which developed a controller that completed the goal task in the substitute simulator. Success rates over time were calculated for varying goal sizes for every combination of parameter settings. The varying goal

point sizes had a radius of either twenty, fourteen, ten or seven centimetres. These various levels of success rate were used to identify how close the substitute real-world robot was able to achieve the desired behaviour.

The best fitness value during controller evolution was periodically recorded and the average fitness of each parameter combination over the 30 trial runs was calculated. The *success rate* for a given parameter combination was its success ratio out of the 30 trial runs. The success rates for the various goal point sizes, along with the average fitness of controllers evaluated for each parameter combination were used to rank how well they performed overall.

Ranking of how well the various parameter combinations performed was done by ranking the success rates according to the seven centimetre goal point radius, then ten, fourteen, twenty and lastly on the average fitness. The top ten best and worst ten parameter combinations could thus be identified through this ranking method. The reason for not simply using average fitnesses for ranking was because the BNS approach could sometimes fail if controller evolution converged towards sub-optimal solutions. Experiments that resulted in a sub-optimal solution would thus result in poor fitness values. These poor fitness values often varied greatly in value and could unfairly skew the average fitness of a parameter combination. A ranking based on ordering parameter combinations according to how well the desired behaviour was achieved was thus chosen.

The diversity of the controller population during controller evolution was measured. The relationship between the diversity and various other properties such as parameter settings, controller success rates and how well the developed simulator was able to predict the substitute real-world fitnesses were also studied. The diversity D for a given controller population was calculated using equation (4.1). The motor speeds and time durations for all controllers were normalized to ensure that there was equal weighting of importance. The normalized left motor speed, right motor speed and time duration of the j^{th} command for the i^{th} controller in the population of controllers are denoted by L_{ij}, R_{ij} and T_{ij} respectively. The controller

population size is denoted by n and the total number of commands for the i^{th} controller is denoted by r_i. The average normalized left motor speed, right motor speed and time duration for the j^{th} command over all controllers is denoted by $\overline{L}_j$, $\overline{R}_j$, and $\overline{T}_j$ respectively.

The diversity provided an indication of the balance between exploration and exploitation of the controller evolution process. A high diversity indicates a large level of exploration where controllers continually explore the search space in order to find new solutions. A low diversity indicates a high level of exploitation where controllers try to improve upon the already explored parts of the search space.

$$D = \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{r_i}|L_{ij} - \overline{L}_j| + |R_{ij} - \overline{R}_j| + |T_{ij} - \overline{T}_j|) \tag{4.1}$$

An obvious requirement for the BNS approach to perform well was the need for controller fitness estimates in simulation to be fairly close to the substitute reality. A fitness error was calculated by computing the absolute difference between a controller's fitness in simulation versus the substitute real-world (in other words a measure of how well the developed SNNs simulated the substitute real-world). At the end of each substitute real-world evaluation, data was collected regarding the controller population diversity, differences between the fitness in simulation and the substitute real-world (referred to as the fitness error) and the success at varying levels of accuracy.

In order to determine how well controllers perform over the lifetime of an experimental run, the relationship between the developing simulator's perceived fitnesses and the substitute real-world fitnesses, over time, were studied. A single experimental run of the BNS approach was conducted using the substitute real-world robot. The parameter settings chosen for the experiment were identical to the real-world validation experiments (Section 4.3). During the ER process, the best controller's fitness in simulation was recorded for every generation. The best controller's corresponding, substitute real-world fitness was also calculated for every generation. The

change between subsequent simulators used for the controller evolution process was also studied.

4.6 Parameter Comparison Results

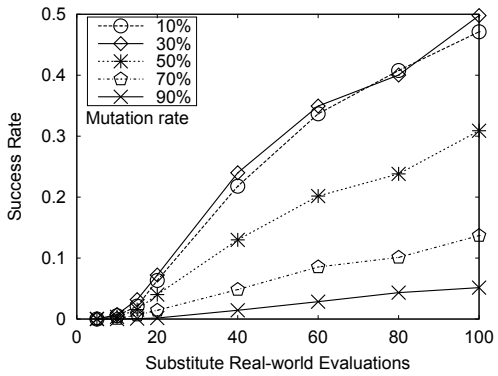
The results of the parameter combination experiments described in the previous section are discussed in this section. Section 4.6.1 discusses success rate results related to all the parameter combination experiments and Section 4.6.2 relates success to the diversity. Section 4.6.3 discusses results related to the best and worst performing parameter combinations.

4.6.1 Success Rate

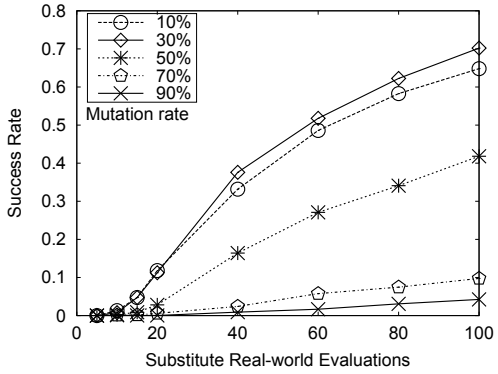
The success rate, over time, for each of the parameter values listed in Table 4.3 over all other parameters is discussed in this section. These success rates were used to determine which parameters were most influential in the success of the BNS approach.

The success rates of the tested controller mutation rates over all experiments are shown in Figure 4.7. The observed success rates demonstrated that the controller mutation rate was the most influential parameter tested. The largest difference between the mutation success rates was approximately 45% for Task 1, 66% for Task 2 and 17% for Task 3. Lower mutation rates generally performed better. Tasks 1 and 2 mostly performed better with a 30% mutation rate, closely followed by 10% whereas Task 3 performed best with a 10% mutation rate.

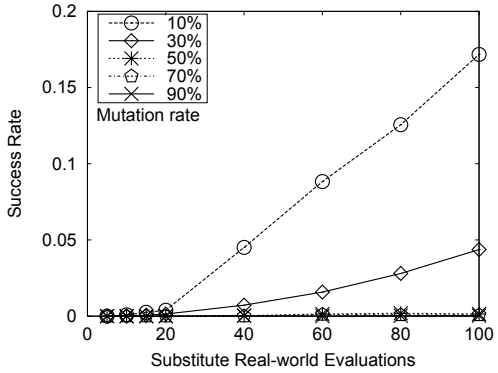
The success rates over time for the tested controller population sizes over all experiments are shown in Figure 4.8. The controller population size was the second most influential parameter tested. The largest difference between the population size success rates was approximately 15% for Task 1, 7% for Task 2 and 4% for Task 3. It was found that larger controller population sizes performed better. During controller evolution, good solutions may be lost due to a changing simulator or a



(a) Task 1

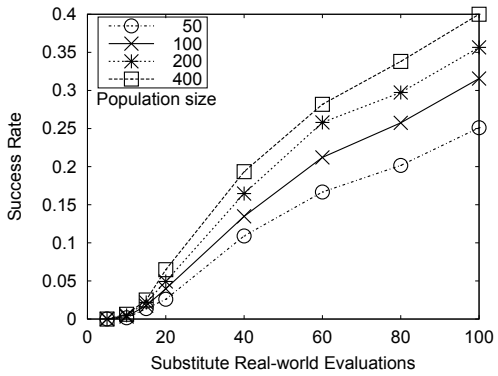


(b) Task 2

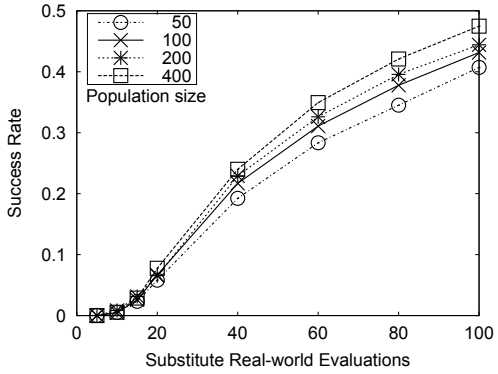


(c) Task 3

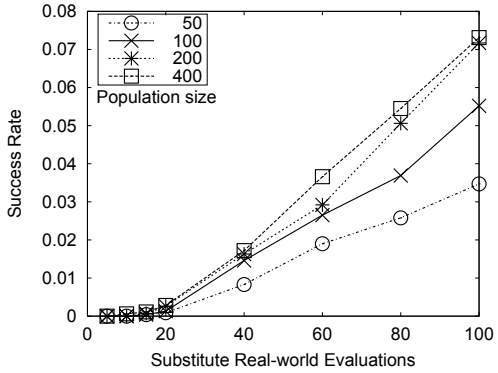
Figure 4.7: Success rate versus number substitute real-world evaluations for the tested controller mutation rates



(a) Task 1



(b) Task 2



(c) Task 3

Figure 4.8: Success rate versus number substitute real-world evaluations for the tested controller population sizes

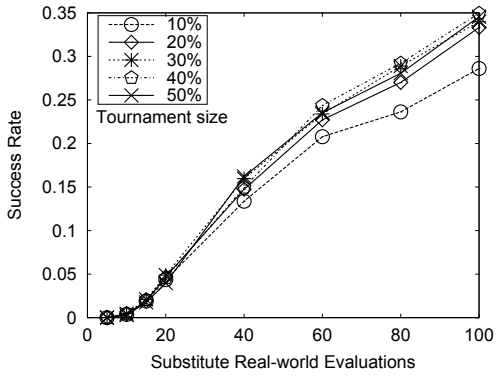
high controller mutation rate. However, a higher population size may contribute towards reducing these issues.

Success rates for the controller and real-world tournament selection sizes are given in Figures 4.9 and 4.10, respectively. The overall result showed that the controller tournament selection size was the third most influential parameter tested and generally performed better for larger values. The largest difference between controller tournament selection size success rates was 6.3% for Task 1, 6.7% for Task 2 and 1% for Task 3. The least influential parameter tested was the real-world tournament selection size which did not appear to have any discernible trend across the different tasks. The largest difference between the success rates of the real-world tournament sizes were 3% for Task 1, 2.5% for Task 2 and 1.7% for Task 3.

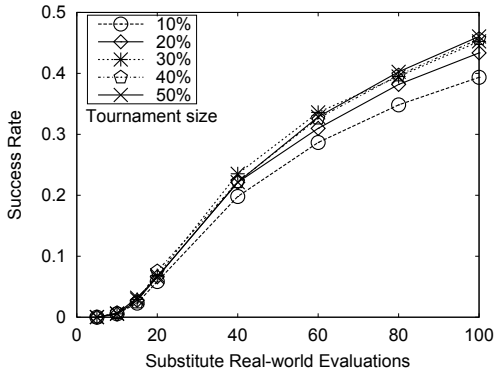
4.6.2 Diversity

To determine how accurately simulators performed in relation to the controller population diversity, a metric called the fitness error was measured. This measures the difference between a controller's fitness in the developed simulator and in the substitute real-world. The average fitness error versus population diversity for each parameter combination after the one hundredth substitute real-world evaluation can be seen in Figure 4.11. The success rate versus diversity of the tested parameter combinations were also investigated. The success rates versus diversity for each parameter combination is shown in Figure 4.12.

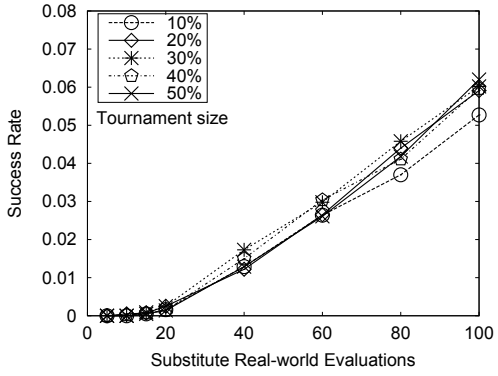
Figures 4.11 and 4.12 indicate that a low diversity had a wider range of fitness errors and success rates. A low diversity could indicate a high level of exploitation where controllers prematurely converged towards sub-optimal solutions. This convergence to sub-optimal solutions allowed the simulator to accurately model the behaviour, however, tasks were often not successfully completed. A very high diversity may indicate that controllers were continually exploring the search space and this resulted in controllers being unable to exploit good solutions.



(a) Task 1

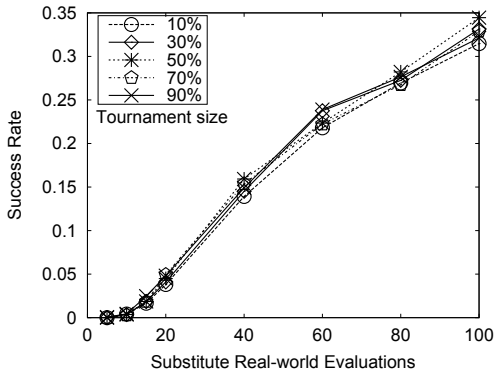


(b) Task 2

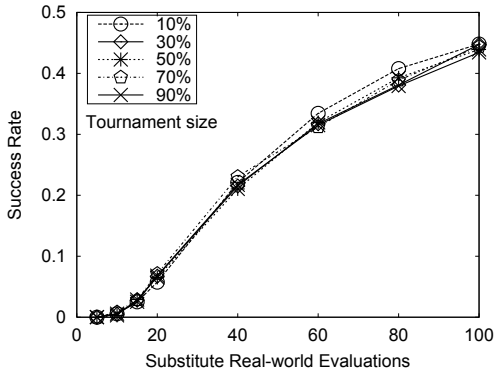


(c) Task 3

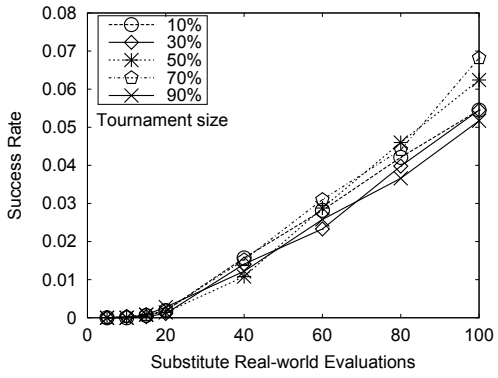
Figure 4.9: Success rate versus number substitute real-world evaluations for the tested controller evolution tournament sizes



(a) Task 1



(b) Task 2



(c) Task 3

Figure 4.10: Success rate versus number substitute real-world evaluations for the tested real-world tournament sizes

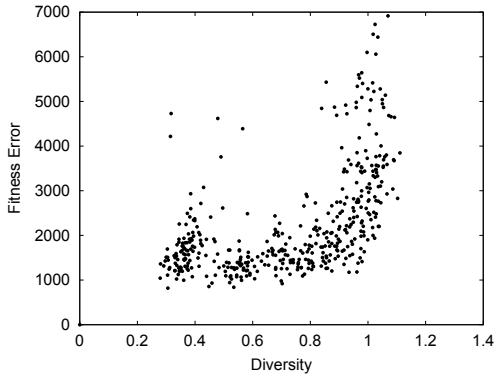
Figure 4.11 shows that the average fitness error generally increased after the diversity passed a certain point. The increase in the fitness error may have been a result of controllers, evaluated in the substitute real-world, not converging towards the desired behaviour. The controllers would thus be less specialised and the training data for the simulator would be less targeted towards the desired behaviour which made simulator training more challenging.

There appeared to be an optimum region of diversity that needed to be maintained to optimise the success of the BNS approach. The optimum diversity regions for Tasks 1 and 2 appeared to be similar (approximately between 0.5 to 0.7), whereas for Task 3 the optimum diversity was approximately between 0.4 to 0.6. These optimum diversity ranges appeared to correspond to where the fitness errors were consistently lower. An obvious reason for this would be that the simulator needed to be sufficiently accurate to evolve better controllers.

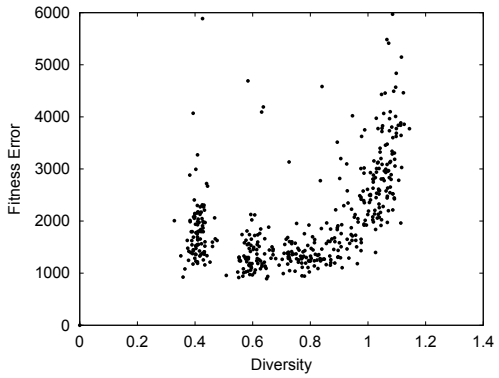
The Task 3 diversity versus success rate (Figure 4.12c) does not appear to have the pattern seen in Figures 4.12a and 4.12b. A possible reason may be the low success rates obtained for Task 3. The Task 3 success rate versus diversity figure was recalculated as shown in Figure 4.12d, with the goal point radius being slightly larger. This made it clear that all success versus diversity plots had a similar pattern. The best performing parameter combination for Task 1 (Figure 4.12a) had a success rate of about 76% and Task 2 (Figure 4.12b) had a success rate of about 90%. Task 3 had a poor success rate for the 7cm goal point radius, where success rates were below 37% (Figure 4.12c). If the goal point radius was relaxed to be 10cm, the best success rate improved to be roughly 79% (Figure 4.12d).

4.6.3 Best and Worst Performing Parameter Combinations

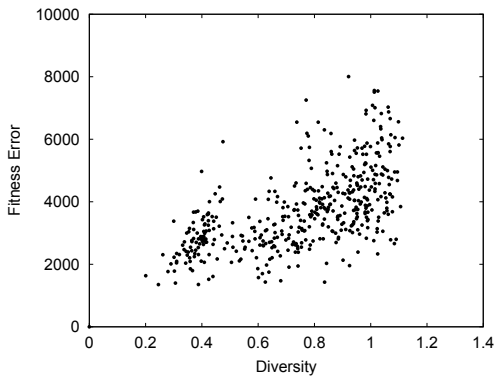
The best and worst performing parameter combinations were compared with each other in order to identify differences, thereby identifying the influential factors. The diversity over time for the top 10 best and worst performing parameter combina-



(a) Task 1

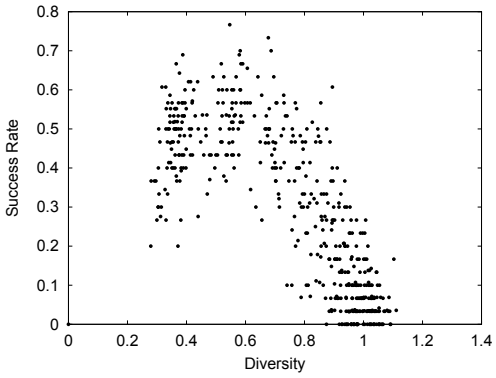


(b) Task 2

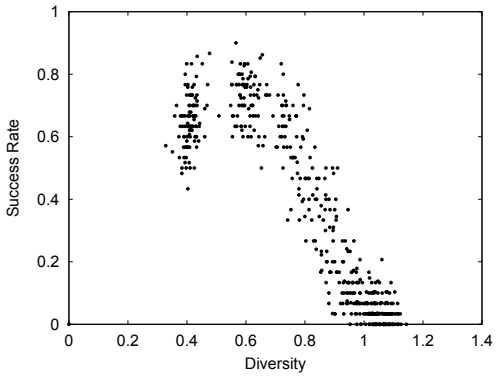


(c) Task 3

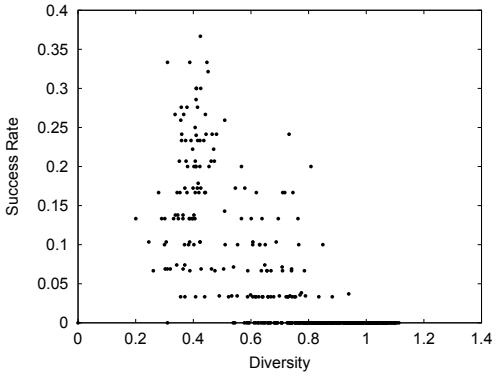
Figure 4.11: Average fitness error versus diversity after 100 substitute real-world evaluations



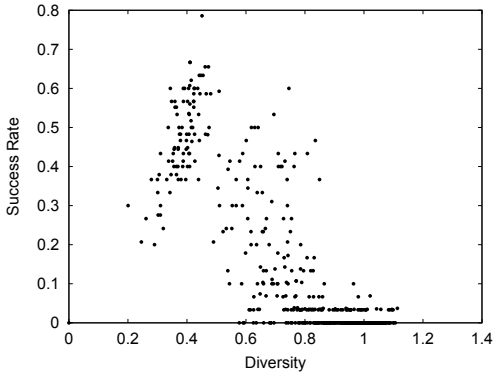
(a) Task 1 (using 7cm goal point radius)



(b) Task 2 (using 7cm goal point radius)



(c) Task 3 (using 7cm goal point radius)

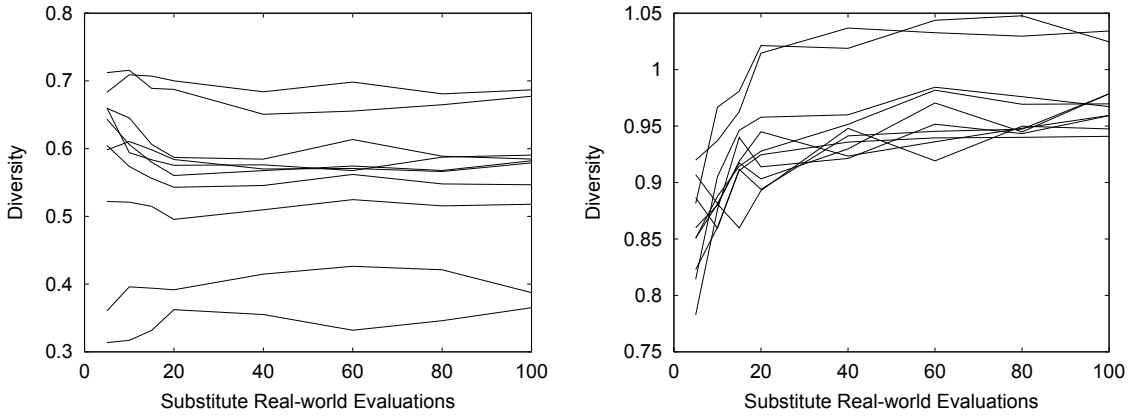


(d) Task 3 (using 10cm goal point radius)

Figure 4.12: Success rate versus diversity after 100 substitute real-world evaluations

tions for all tasks can be seen in Figures 4.13 to 4.15. The diversity for the first few substitute real-world evaluations of the BNS approach was usually the highest. However, there was little point in analysing how the diversity behaved before the fifth substitute real-world evaluation, as controller evaluations were almost random. For this reason, figures do not present the diversity before the fifth substitute real-world evaluation.

The top 10 best performing parameter combinations had a lower diversity when compared to the worst performers for each task. A higher diversity may have been due to controllers having a higher mutation rate which could have resulted in controller evolution being unable to exploit good solutions. Controllers that are mutated



(a) Top 10 parameter combinations

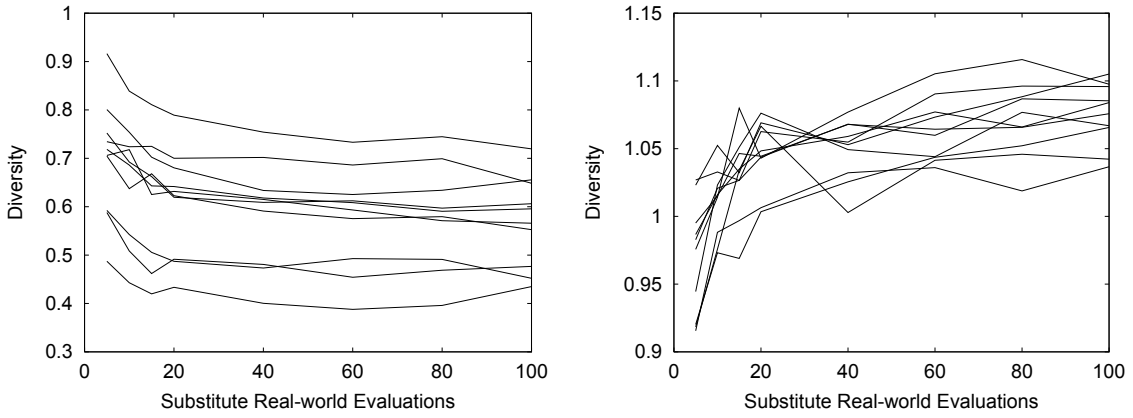
(b) Worst 10 parameter combinations

Figure 4.13: Diversity versus number of substitute real-world evaluations for Task 1

too much, together with the issue of the SNNs continually changing may have contributed to higher diversities performing poorly. Most of the 10 worst performers had mutation rates that were 70% or above for all tasks. The vast majority of the 10 best performers had mutation rates of 30% or less.

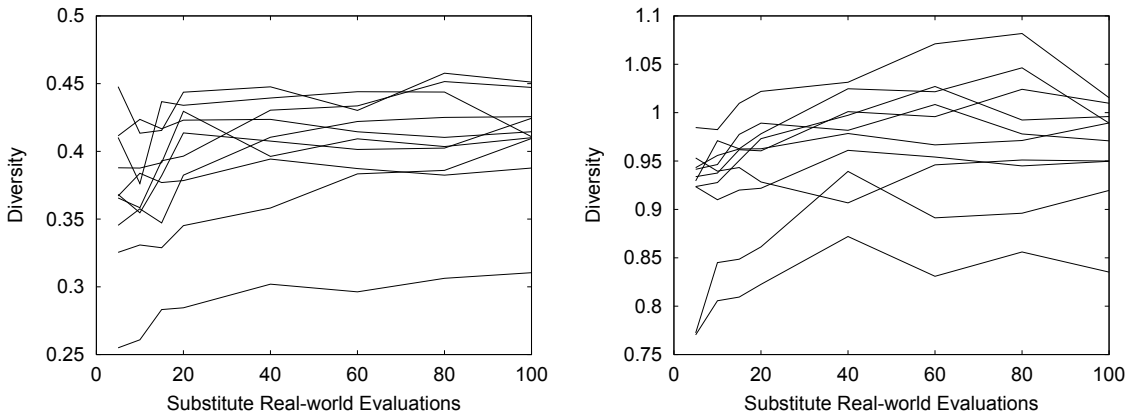
In the previous section, the optimum diversity range was an estimation based on success rates and fitness errors. Six of the ten for Task 2 and eight out of ten for Tasks 1 and 3 were within their estimated optimum diversity ranges. The diversities of the best performers were fairly constant after the twentieth substitute real-world evaluation for the first two tasks and after the fortieth substitute real-world evaluation for the third task (Figures 4.13a to 4.15b). The diversity of the worst performers tended to increase over time for all tasks.

An interesting observation emerged where the first two tasks had a larger range of diversities for the best parameter combinations when compared with the worst performers. The reverse was seen for Task 3, where the worst performers had a larger range of diversity. All the worst performers for Task 3 had a 0% success rate for all levels of success. The ranking, therefore, had to be based on greatly varying fitness estimates which resulted in a wider diversity range. The higher diversities for worst performers indicated that a saturation point was reached where the diversity could



(a) Top 10 parameter combinations (b) Worst 10 parameter combinations

Figure 4.14: Diversity versus number of substitute real-world evaluations for Task 2



(a) Top 10 parameter combinations (b) Worst 10 parameter combinations

Figure 4.15: Diversity versus number of substitute real-world evaluations for Task 3

not become much higher. These saturation ranges were confirmed by comparing them with the maximum diversities obtained in Figures 4.11 and 4.12.

A metric was chosen that indicated how rapidly the BNS approach converged towards the desired behaviour. This potential success indicator was the success rate for a given task with the goal point radius size being redefined to be twenty centimetres (for example, in Task 1, the robot would at least be moving through the correct quadrants of the xy-plane). The convergence towards the target behaviour

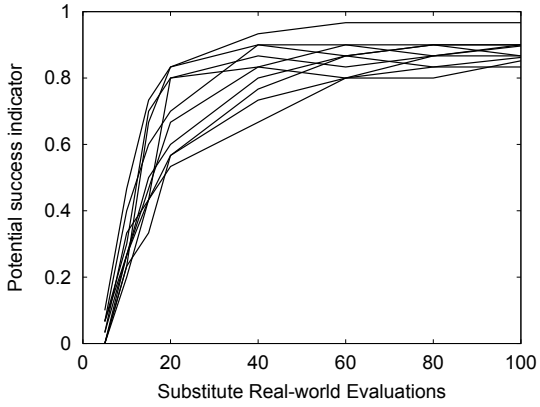
over substitute real-world evaluations for the top 10 best performing parameter combinations can be observed in Figure 4.16. Task 2 generally converged towards the desired behaviour the soonest, Task 1 converged more slowly and Task 3 converged much later. Task 3 clearly struggled to converge towards the desired behaviour which could indicate that solutions often converged to sub-optimal solutions or good solutions were frequently lost or evolution may have been stopped too early.

Figure 4.17a illustrates the substitute real-world's observed fitnesses and the simulator's assigned fitnesses over the duration of a single experimental run of Task 2. The experiment terminated after thirty substitute, real-world controller evaluations. The initial fitness assignments were volatile and inaccurate, but, over time, stabilized as the simulator became more accurate. The substitute real-world fitness was mostly found to be less than the simulator's perceived fitness. A small drop in fitness was occasionally observed which corresponded to changes to the simulator.

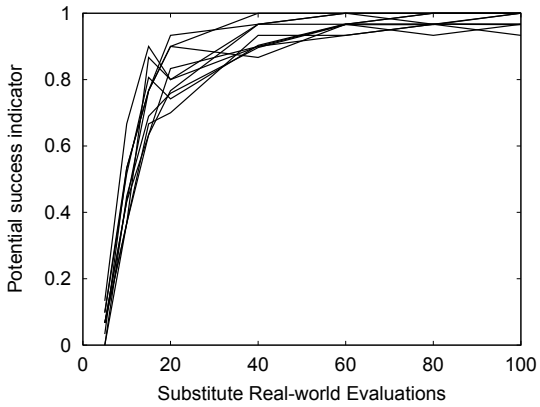
A set of ten randomly generated controllers was created before the experimental run and each controller contained five commands that were generated from a uniform distribution. These controllers were evaluated periodically over time and the final positions reached were noted. Figure 4.17b represents how close these final positions were in simulation to the substitute real-world. The position difference varied significantly during the early stages of the experiment and later stabilized. Each dot in Figure 4.17c represents the total difference between the final positions of the generated controllers for two successive generations of the developing simulator. This demonstrated that there were large changes between successive simulators which gradually stabilized to become less severe. The time taken to perform successive iterations increases as more training data is added.

4.7 Conclusions

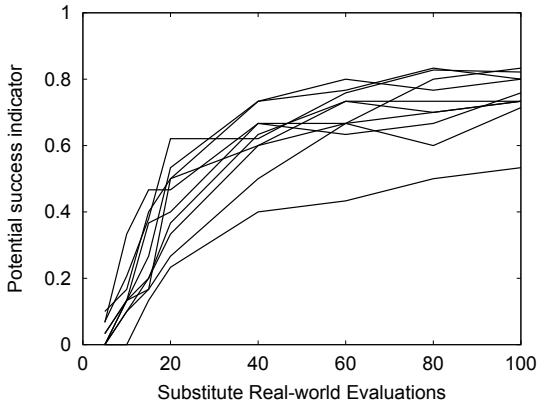
A vital goal in ER worth exploring has been shown to be the automatic, real-time creation of controllers and simulators with minimal human intervention or specialised



(a) Task 1

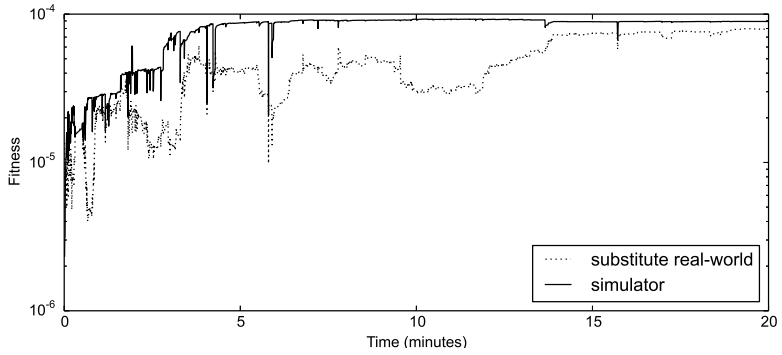


(b) Task 2

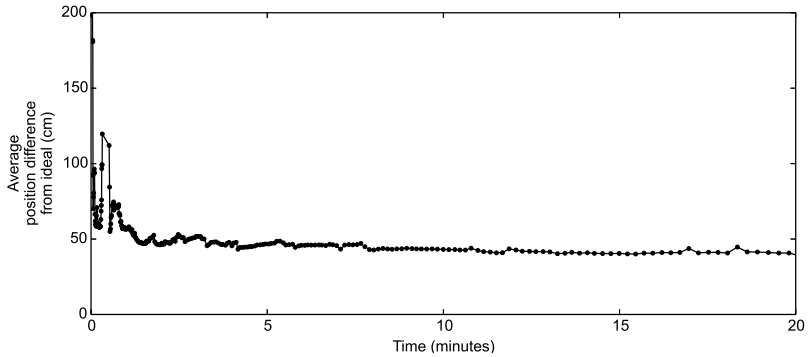


(c) Task 3

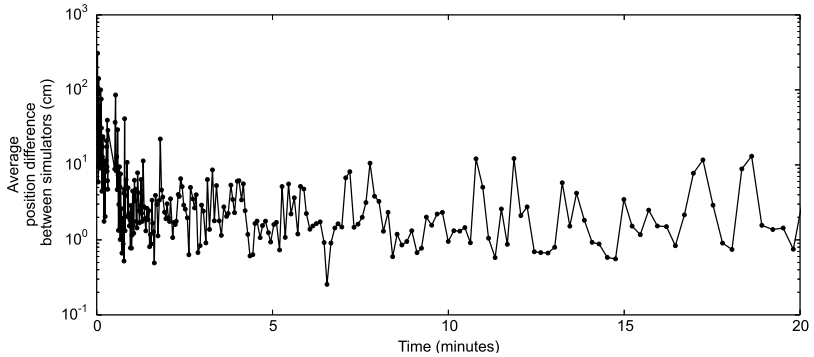
Figure 4.16: Potential success indicator versus number of substitute real-world evaluations for the top 10 parameter combinations



(a) Fitness over time



(b) Position difference over time



(c) Change in position displacement over time

Figure 4.17: Experimental run over time

knowledge. SNNs have been shown to provide a viable alternative to traditional simulators for this goal. Many traditional approaches to simulator development are complex and require specialised knowledge while SNNs are relatively simple to develop. Bi-directional approaches to controller and simulator development have also shown much promise in automating simulator development.

The BNS approach demonstrated accurate modelling of robot behaviour and also produced successful controllers for performing trajectory planning tasks using a robot. This research focuses on developing simulators that are specialised according to the required behaviour. The specialisation was due to the evaluation of task specific controllers for behavioural data, as opposed to the development of a more general simulator. The use of task specific controllers attempted to reduce the number of controller evaluations needed to train the simulator, thereby speeding up the ER process.

The controller population size and controller mutation rate were shown to be the most influential parameters in controlling the success of the BNS approach. However, there was no guarantee of a successful solution. Controller evolution may converge to sub-optimal solutions or good solutions may be lost due to a changing simulator. Higher population sizes can result in fewer instances where good solutions were lost due to a changing simulator. Lower mutation rates were shown to perform better than those with higher values which could be because higher values did not allow controllers to exploit important features.

The first prototype developed in this work demonstrated that a bi-directional approach to controller and SNN development during the ER process is indeed viable. In order to validate that the BNS approach is scalable to more complex robots, the next chapter demonstrates the BNS approach on a snake-like robot.

Chapter 5

SNAKE ROBOT PROTOTYPE

5.1 Introduction

This chapter describes the experimental work performed on a snake-like robot. The background relating to snake locomotion is discussed in Section 5.2. The procedures used for investigating the BNS approach are discussed in Section 5.3. Due to a lack of previous work done on SNNs for snake-like robots, an initial investigation was conducted into the use of pre-computed SNNs for a snake-like robot (Section 5.4) and results relating to the experimental work are discussed (Section 5.5). After the viability of pre-computed SNNs was demonstrated, work was conducted for validating the BNS approach on a real-world, snake-like robot (Section 5.6) and the results of the experimental work are presented (Section 5.7). Parameter comparison experiments were conducted and the related results are discussed in Sections 5.8 and 5.9, respectively. Finally, conclusions are drawn based on the results of this chapter (Section 5.10).

5.2 Snake Locomotion

Biologically inspired snake-locomotion modes can be broadly classified into lateral undulation, slide-pushing, rectilinear motion, concertina, side-winding and various

other forms [Dowling, 1996]. Lateral undulation (Figure 5.1) is the most common form where all parts of the snake body move in a wave-like pattern [Dowling, 1996]. Lateral undulation is not suited to smooth, low friction surfaces [Shmakov, 2006].

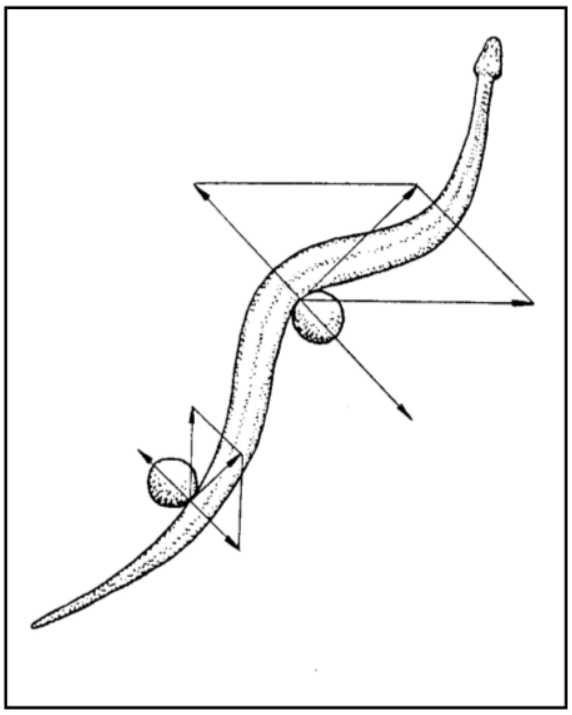


Figure 5.1: Lateral undulation of snake [Dowling, 1996]

Friction is a very important factor in snake locomotion. Biological snakes have directional friction where the snake's body has less friction moving forwards when compared to moving backwards due to the direction of the overlapping scales [Hu, Nirody, Scott, and Shelley, 2009]. Many snake-like robots use wheels to achieve similar frictional properties.

Side-winding locomotion (Figure 5.2) has a sine-like wave while maintaining only two static points of contact with the ground at any time. Side-winding is more suited to low friction surfaces [Shmakov, 2006]. There are also non-biologically inspired locomotion modes, namely rolling, where the snake rolls side over side and flapping motions where the robot flaps both of its ends across the ground [Dowling, 1996].

Snake-like robots are constructed by chaining together several independent ac-

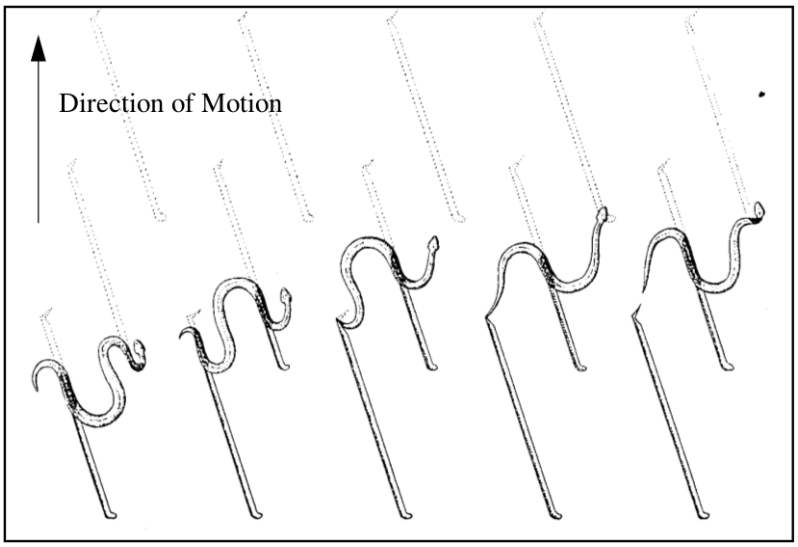


Figure 5.2: Side-winding locomotion of snake [Dowling, 1996]

tuators, where each actuator commonly has one degree of freedom [Kamimura, Kurokawa, Yoshida, Murata, Tomita, and Kokaji, 2005; Melo, Hernandez, and Gonzalez, 2012a; Melo, Paez, and Parra, 2012b]. These snake robots are capable of both biological and non-biologically inspired locomotion modes.

Snake robots typically consist of many active joints, providing much redundancy and versatility. These robots are capable of operating over various surfaces unsuitable for wheeled motion, such as strafing laterally, swimming, climbing inside and outside of pipes, climbing stairs and various others [Melo *et al.*, 2012a].

Parametrized equations can be used to generate the above mentioned locomotion modes which generate the appropriate joint angles on the robot. Melo *et al.* [2012a] made use of parametrized equations (5.1) and (5.2) which are based on sinusoidal motion on two axes. These equations define $\phi(n,t)$ as the angle of the n^{th} joint at discrete time step t. Snake joints were numbered, starting at zero from front to back, with the evenly numbered joints moving the snake laterally and the odd ones moving the snake vertically (Figure 5.3).

$$\phi(n,t) = \begin{cases} O_{lateral} + A_{lateral} \cdot sin(\theta + \alpha), & n \text{ is even} \\ O_{vertical} + A_{vertical} \cdot sin(\theta), & \text{otherwise} \end{cases} \tag{5.1}$$

$$\theta = \omega n + \gamma t \tag{5.2}$$

If the snake rolled on its side, then the evenly numbered joints moved vertically and oddly numbered joints moved laterally which similarly switched the angles used by equation (5.1). The terms $O_{lateral}$ and $O_{vertical}$ are the offsets of the lateral and vertical joints respectively and define the central angle value of their respective waves. Terms $A_{lateral}$ and $A_{vertical}$ refer to the wave amplitudes of the lateral and vertical joint angles, respectively. The parameters ω and γ are respectively the spatial and temporal components of the sine wave moving through the robot. The α term is the phase offset between the lateral and vertical sine waves.

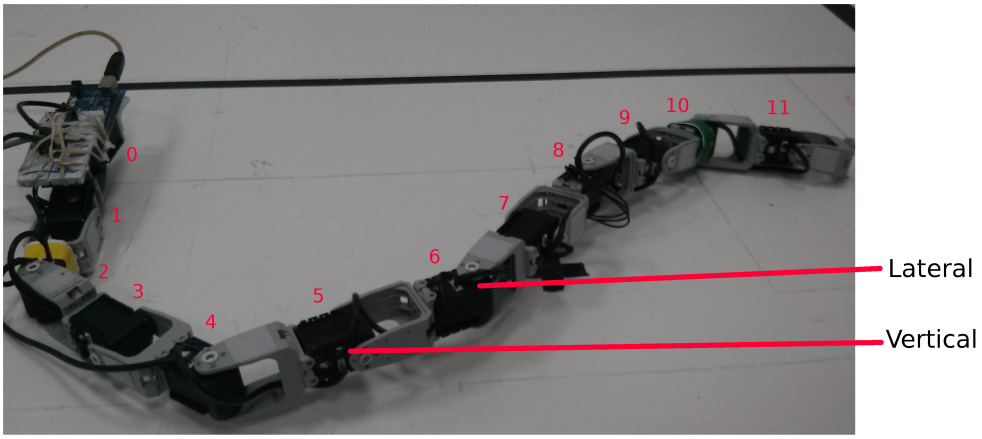


Figure 5.3: Snake Robot

5.3 Experimental Procedure

The snake prototype developed in this work is a first-time investigation into the use of SNNs on a complex robot with a high number of degrees of freedom. Therefore controllers and SNNs were designed with a few simplifying assumptions. The following sections provide details regarding the experimental work and procedures that

were followed. Section 5.3.1 discusses the hardware and data capturing techniques. Details of the controller used are discussed in Section 5.3.2 and the developed simulator is described in Section 5.3.3.

5.3.1 Hardware and Data Capturing

A custom designed, snake-like robot was developed for the experimental work. Similarly constructed snake-like robots have also been used by other researchers [Melo *et al.*, 2012b; Murata and Kurokawa, 2007]. This robot was chosen due to its simple construction and relatively low cost. The morphology of the robot used for experimental work is shown in Figure 5.4. The robot consisted of an array of twelve Dynamixel AX-12 servo motors joined together with links. Evenly numbered joints moved the robot laterally while the odd joints moved the robot vertically. The length of the snake robot is 114cm and its width and height are 5cm. The servos are controlled by an Arduino Mega micro-controller which received joint angles from a computer using serial communication.

The servos are powered by using a tethered connection to the robot. A tethered approach was chosen due to the extra weight batteries would add and to eliminate the need for monitoring battery-power levels. Near each end of the robot was a yellow or green tracking marker which is used for camera-based tracking.

No snake-like skins or mechanisms such as wheels were used to allow for the directional friction required for forward locomotion seen in many biological snakes (Section 5.2). The robot consequently had difficulty with forward locomotion and had to either side-wind, strafe laterally, perform helix-like rolling or flapping motions. The robot was not fitted with any sensors.

A roof-mounted camera was used to track robot behaviours, namely the change in position of the tracked markers on the robot for a given behaviour. A camera-based tracking approach was chosen over manual data acquisition methods in order to speed up the process and eliminate human error. An open source, computer vision

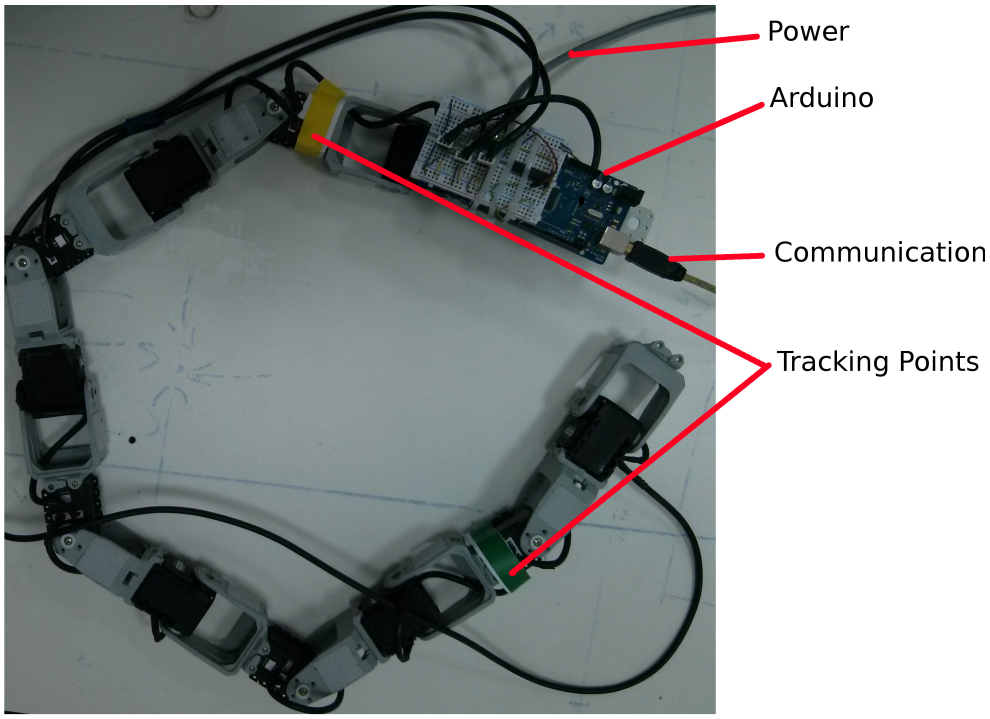


Figure 5.4: Robot Morphology

library called OpenCV [OpenCV, 2015] was used for the camera-based tracking software.

Images from the tracking camera were used to locate the pixel coordinates of the tracked markers. The centre of each marker was determined and converted into real-world coordinates. Calibration techniques were employed and distortions were removed from the captured images before use. The automated tracking process was not used to identify if the robot tipped over, rolled, collided with itself or if the torque limits were reached, therefore these behaviours were manually noted during data acquisition.

5.3.2 Controllers

Because this investigation was an initial proof-of-concept prototype, the trajectory planning tasks were deliberately chosen to be simple in order to investigate the

viability of the approach. The robot had to visit square goal regions in a specific order. The required navigational behaviour is shown in Figures 5.11 to 5.13 and Figures 5.16 to 5.21. Square regions were 40cm wide and were placed in two or three quadrants of the operating surface and placed 20cm apart from each other. The robot was placed with the yellow marker on the origin and facing eastwards. The task was considered completed when the robot traversed all the blocks in a specific order. During the ER process, controllers were evaluated in simulation which generated a list of points that represented the simulated path. The fitness values assigned to each controller were determined by using Algorithm 3. This algorithm took as input the simulated path and the list of goal regions. The fitness was calculated based on the number of goal regions reached in their specified order. The robot was judged to have reached a given goal region if the midpoint of the tracked markers moved within the goal region. A controller's fitness was penalized for containing positions outside the bounds of the working surface. A cycle was defined to be unstable if any of the robot's motors reached a torque limit, or if the robot collided with itself or the robot did not remain upright during the cycle. If a position was reached from an unstable cycle, fitness was also penalized. Fitness was further penalized for each goal region not reached.

A set of twelve joint angles represented a single command. When a command was sent to the robot, all of the robot's joints simultaneously positioned themselves to the assigned angles. A cycle consisted of thirteen sequential commands which, when evaluated, started and ended on the same joint angle set. During the evaluation of a cycle, commands were sent sequentially to the robot and upon reaching all the assigned joint angles of the given command, the robot moved on to the next command. The controllers developed for the snake robot were for open-loop navigation. A controller consisted of a sequential list of ten different cycle parameter settings for equations (5.1) and (5.2) (Figure 5.5). Each cycle in the list was evaluated sequentially and could be repeated up to four times when evaluated.

Controllers were evaluated on the real-world robot according to the cycles re-

Algorithm 3: Snake Controller fitness evaluation

Data:

$positions \leftarrow$ Simulated path list

$goalpoints \leftarrow$ List of goal regions

Result: Fitness of controller

$sum \leftarrow 0$

$curGP \leftarrow 0$ ▷ Current goal point

$penalty \leftarrow$ a large constant

for *each position in positions* **do**

  if *position out of bounds* **then**

    $sum \leftarrow sum + penalty$

  if *position reached* $goalpoints[curGP]$ **then**

    $curGP \leftarrow curGP + 1$

    if *all goals reached* **then**

      break

  if *position reached from failed cycle* **then**

    $sum \leftarrow sum + penalty$

$missedGoalpoints \leftarrow length(goalpoints) - curGP - 1$

$sum \leftarrow sum +$ distance to $goalpoints[curGP]$

$sum \leftarrow sum + penalty \times missedGoalpoints$

$fitness \leftarrow 1/sum$

return *fitness*

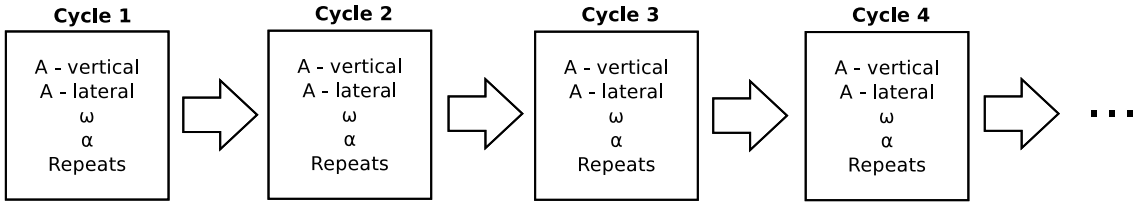


Figure 5.5: Controller morphology

quired to complete the goal task in simulation and excess cycles were ignored during real-world evaluations. The most successful controllers require far fewer than ten unique cycles to complete the tasks given in this work. Cycles were generated by using equations (5.1) and (5.2). The γ term determined the number of commands contained in a given cycle. The set of joint angles for all cycles had to begin and end on the same set of angles which is exactly one period of the sine wave. The time-steps of t for every cycle went from zero to the number of commands per cycle plus one. The commands per cycle were fixed at thirteen which required that the γ term remained constant at $2\pi/12$. The offsets $O_{lateral}$ and $O_{vertical}$ determined the central angle value for the wave which was assumed to remain constant at zero. Non-zero offsets typically help to steer the robot in a particular direction. Amplitudes $A_{lateral}$ and $A_{vertical}$ had a range of between 0 and $\pi/2$. The α and ω parameter values ranged between 0 to 2π. These ranges were chosen based on experimental work. The reason why the number of commands per cycle, the offsets and γ terms were chosen to remain constant was to reduce the controller search space which in turn reduced the amount of training data the simulator required to perform adequately.

The vertical amplitude $A_{vertical}$ was chosen so that it was less than the lateral amplitude $A_{lateral}$ in order to reduce the number of cycles that failed to remain upright. Even with this restriction in place, many parameter sets could result in the robot's tipping over or rolling. Ensuring that the robot always remained upright and stable during evaluations was important for the simulator predictions. The simulator was trained to predict only the behaviours of stable cycles.

The angular velocity of the robot's joint movements was kept constant. The

angular velocity was experimentally chosen to reduce slippage on the smooth operating surface. Depending on the locomotion mode, the degree of slippage varied greatly. Due to the flexibility of equation (5.1), many different locomotion modes were possible, even modes that resulted in collision or caused joint torque limits to be reached.

5.3.3 Simulator

Equation (5.1) was used to generate a sequence of commands that perform cyclical behaviours when evaluated on the robot. The robot's change in position after the evaluation of a given cycle depends on the parameter values used by equations (5.1) and (5.2). The developed SNNs took as input those parameter settings and subsequently predicted the change in the robot's position on the planar operating surface at the end of the given cycle. The simulator also simulated whether cycles would fail and cause the robot to tip over, roll, reach torque limits or collide with itself.

Because this research was a first-time investigation into the viability of using SNNs to simulate a complex, snake-like robot and, in order to keep complexity to a minimum, a few simplifications were applied. The simulator predicted the change in the robot's position for a given cycle, however, the change in the position of the robot when transitioning between different cycles was ignored. The change in the robot's position between cycles was usually small enough to ignore. The distance between tracked markers for each cycle was not predicted. The distance between the tracked markers was assumed initially to be 65cm at the start of every controller evaluation. Distances between tracked markers could increase or decrease based on simulator predictions and the distance between markers could become unrealistically large or small during a controller evaluation.

SNNs were trained to predict the change in the position of the tracked markers on the robot for any given cycle. The robot had two local coordinate systems with

the origins located at the centre of each tracked marker and y-directions taken from the yellow to green tracked markers. The change in the x and y-coordinates for the yellow tracked marker for a given cycle was represented by Δx_1 and Δy_1, respectively. Similarly, the changes in the x and y-coordinates for the green marker were represented by Δx_2 and Δy_2, respectively. The simulator also predicted whether or not cycles remained stable.

The simulator developed in this study consisted of five separate Feed-Forward Neural Networks (FFNNs) (Figure 5.6), one for each of the x-and y-directions for each tracked marker and another to predict failed cycles. A previous study determined that separate FFNNs could produce more accurate results than a single FFNN [Pretorius *et al.*, 2009]. Each FFNN took as input the values of $A_{lateral}$, $A_{vertical}$, ω and α that were used by equation (5.1) to generate the cycle. The offset and γ terms were kept constant and not used as inputs. Sigmoid activation functions were used for all FFNN neurons and each FFNN had a single hidden layer of 100 neurons. The optimal number of hidden neurons was determined experimentally. The sigmoid function was chosen because it was experimentally determined to improve training efficiency. In order to evolve robust controllers that were able to cross the reality-gap, noise was injected into the ER process during controller evaluations.

5.4 Pre-computed Snake Robot Simulator

The previous parameter comparison experiments (Section 4.5) that were conducted on the Khepera robot were only possible due to the development of a pre-computed simulator from previous research. This was a first-time investigation into the use of SNNs on a snake-like robot. No pre-computed snake simulator was available for conducting the parameter comparison investigations for the snake robot. Additionally, an initial investigation into the viability of simulating snake-like robot locomotion by using SNNs needed to be conducted. Pre-computed SNNs for a snake-like robot were developed and validated before the BNS approach was attempted. Section

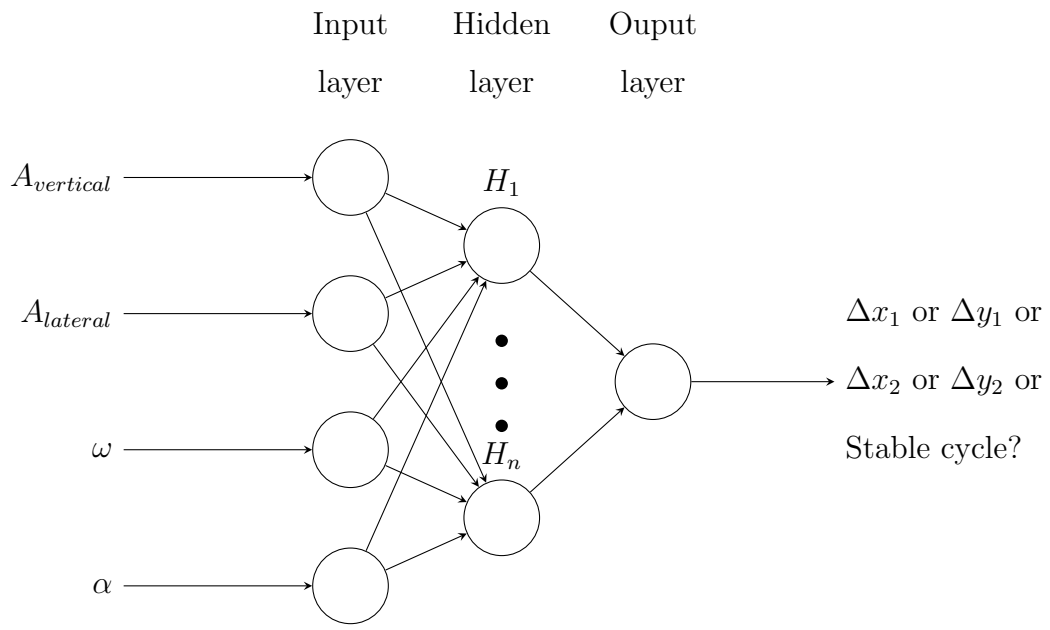


Figure 5.6: Simulator Neural Networks of the snake robot

5.4.1 describes the experimental procedure followed to acquire and use behavioural data to train the pre-computed simulator and produce controllers. The resulting controllers were validated by performing trajectory planning tasks on the real-world robot (Section 5.4.2).

5.4.1 Experimental Procedure for Pre-computed Simulator

Equation (5.1) was used to generate a sequence of commands that perform cyclical behaviours when evaluated on the robot. The robot's change in position after the evaluation of a given cycle depended on the parameter settings used by equations (5.1) and (5.2). The creation of SNNs and controllers using the ER process was achieved as follows:

1. Parameter settings for equations (5.1) and (5.2) are randomly generated using a uniform distribution and are used to generate cycles that are performed on the real-world robot. As a result of these commands, the robot moves around the environment and data is collected by using motion tracking techniques.

2. When sufficient data has been collected, it is used to train SNNs to predict the real-world robot's behaviours. Data collection is concluded when the trained SNNs are able to evolve adequately transferable controllers that complete the goal task.

3. The trained SNNs are used to determine the fitness of candidate controllers during controller evolution.

4. At the end of the ER process, the fittest controller in simulation is validated by using the real-world robot.

Training data was experimentally acquired from 400 randomly generated cycles. This training data was used to train an initial simulator and the worst 20 training data cycles that had the largest difference between their expected versus simulated displacements were subsequently selected for each of the SNNs. These poor performers were re-evaluated and each one was used to generate three additional cycles with noise, and generated 400 additional training data cycles. The SNNs were then re-trained by utilizing all training data. These SNNs were used to evolve controllers during the ER process and twenty of the worst performing cycles from these controllers were taken and used to generate a further 100 training data patterns. The final SNNs were then trained by using all available training data generated.

SNNs were trained using Resilient Backpropagation for 12000 iterations or just before over-fitting occurred. The sizes of the training data set and verification data of the cycles that remained stable were 720 and 76, respectively. The sizes of the training data and verification data sets that included unstable cycles were 818 and 82, respectively. The data for the stable cycles was used to train the SNNs to predict the changes in the x and y position of both tracked markers for a given cycle. The data for the stable and unstable cycles was used to train another SNN to predict whether a given cycle would be stable or unstable.

In order to evolve robust controllers that were able to cross the reality-gap, noise was injected into the ER process during controller evaluations. Controllers were

evaluated ten times in simulation and the average fitness was used. The distribution used for noise injection was Gaussian with a mean of zero and standard deviations of 10cm and 5cm for the Δx_1, Δx_2 and Δy_1, Δy_2 displacements respectively. These standard deviations were based on the observed training data errors.

A controller consisted of a sequential list of four different parameter settings that were used to generate cycles using equations (5.1) and (5.2). Each cycle in the list was repeated up to four iterations when evaluated. A single controller could thus consist of between four to sixteen cycles.

5.4.2 Validation Experiments

Controllers consisted of encoded parameters for the variable terms in equations (5.1) and (5.2). The controller evolution settings are given in Table 5.1.

Table 5.1: Parameters for controller evolution

Population Size	400
Initialization	Random from a uniform distribution
Selection	Tournament (size 20)
Cross-over Method	Uniform
Mutation Probability	10%
Mutation Method	Random Component Perturbation

An initial population of 400 controllers was generated randomly from a uniform distribution. During uniform cross-over operations, two parents were selected by using tournament selection (using a tournament size of 20 parents). Child controllers consisted of approximately 20% genetic material from one parent and 80% from the other. A 80/20 split in the genetic material was assumed to help reduce the loss of

known, good solutions during the cross-over process, but later experimental work indicated that there was little difference compared to a 50% split. There was a 10% probability that controller parameter values were mutated by an amount that was uniformly chosen in the range [-0.2, 0.2] for the α and ω terms and in the range of [-20, 20] for the amplitudes. These settings were based on the results of the Khepera parameter combination experiments and prior experimental work. The ER process proceeded until the fittest controller was sufficiently able to complete the task in simulation. Upon completion, the fittest controller was evaluated on the real-world robot.

The accuracy and viability of using SNNs for the snake-like robot was demonstrated through a set of experiments. Controllers were developed to allow the robot to navigate through three square goal regions. The robot was placed just below the first goal region and had to traverse all the blocks in a specific order. The following section covers the results of the experimental work.

5.5 Pre-computed Snake Robot Simulator Results

The accuracy of the pre-computed simulator is investigated in Section 5.5.1. Controllers were evolved using the pre-computed simulator and validated on a real-world robot (Section 5.5.2). Lastly, these results are discussed in Section 5.5.3.

5.5.1 Simulator Accuracy

Figures 5.7 to 5.10 demonstrate the predicted versus expected change in the position of the tracked markers for the verification data set. The predicted change in position was determined by the simulator and the expected change in position was obtained through real-world evaluations. Every dot represents the simulated versus real-world change in position of the robot for the specific coordinate axis and for the given verification pattern. The verification data set was not presented to the simulator during training and was therefore used to determine the accuracy and generalisation

abilities of the trained simulator.

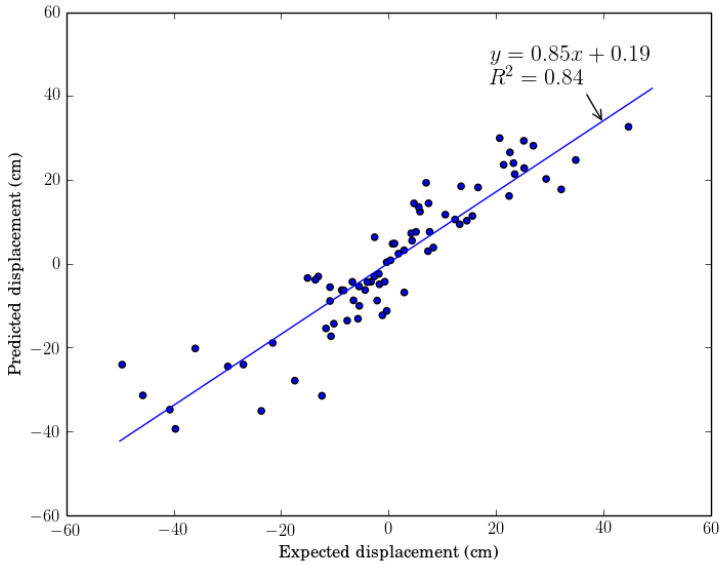


Figure 5.7: Predicted versus expected displacement in the x-direction for the yellow tracking marker

For Figures 5.7 to 5.10, linear regression lines were fitted. All the y-intercepts of the regression lines were close to zero. The R-squared values for Figures 5.7 and 5.9 were 0.84 and 0.75, respectively. This indicated that the simulator could adequately model movement along the x-direction of the robot. The regression line slopes were 0.85 and 0.9 for Figures 5.7 and 5.9 respectively and both y-intercepts close to zero, which indicated that the simulator modelled the real-world fairly well. The R-squared values, 0.17 and 0.05 for Figures 5.8 and 5.10 respectively, indicted that the simulator could not adequately model movement along the y-direction of the robot. The slopes for Figures 5.8 and 5.10 were 0.28 and 0.25 respectively, which indicated that the simulator and real-world did not closely relate for those movements. The sum of squares of errors for the x-direction for the yellow and green tracked markers were 3548 and 15912 respectively which indicated that the simulator could more accurately predict the movement of the yellow tracked marker.

The simulator was trained to predict the success or failure of any given cycle.

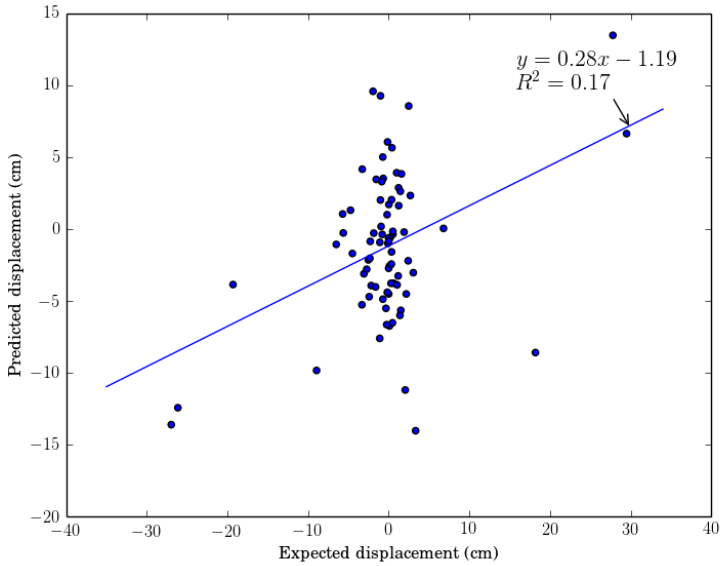


Figure 5.8: Predicted versus expected displacement in the y-direction for the yellow tracking marker

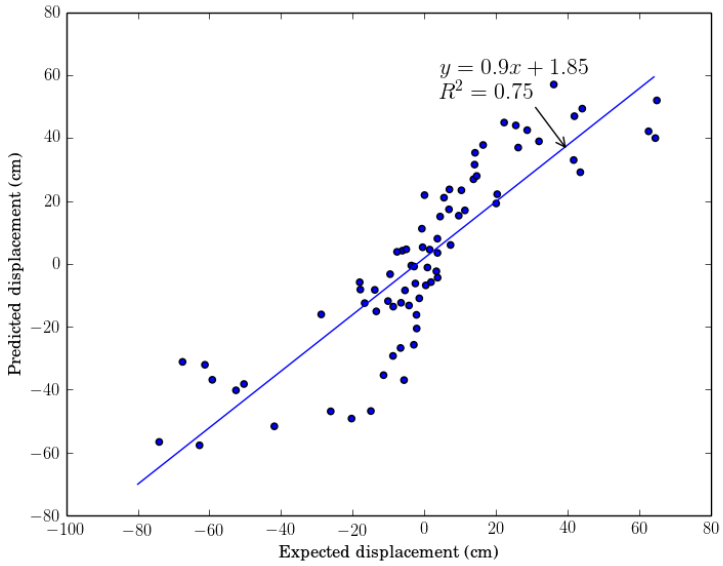


Figure 5.9: Predicted versus expected displacement in the x-direction for the green tracking marker

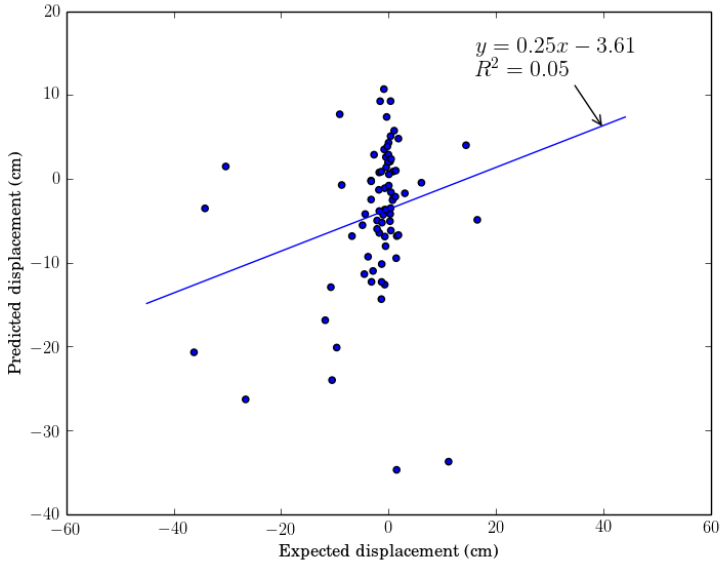


Figure 5.10: Predicted versus expected displacement in the y-direction for the green tracking marker

The patterns used to verify the success or failure of a cycle contained 19 failed cycles and 63 successful cycles. The percentage of verification patterns that the simulator correctly predicted to succeed or fail was 85%. The simulator relied greatly on only predicting behaviours of successful cycles which meant that fitness estimations were inaccurate for controllers that contained any failed cycles. Out of the 63 successful cycles in the verification set, the simulator correctly predicted that 94% were successful. For the verification set's 19 failed cycles, the simulator correctly predicted 58% would fail. This meant that the simulator could consistently identify successful cycles but was less capable of identifying failed cycles.

Predictions for the robot's movement in the y-direction (Figures 5.8 and 5.10) were a great deal less accurate than those along the robot's x-direction (Figures 5.7 and 5.9). The expected values for the robot's y-direction as shown in Figures 5.8 and 5.10 tended to be close to zero and the range of predicted values were also closer to zero compared to the x-direction (Figures 5.7 and 5.9). This was possibly due to the smooth operating surface and lack of directional friction properties seen

in biological snakes or snake-like robots using wheels. These results can potentially be explained by assuming that much of the movement in the y-direction was close to random and that the main strategy used by the robot for locomotion was in the x-direction.

The results provided a reliable indication as to the accuracy of the developed simulator and whether enough training patterns had been collected. Predicting the robot's change in positions is indeed possible, which is significant, considering the complexity of the robotic system.

5.5.2 Trajectory Planning Results

The ideal paths obtained by validation controllers in simulation were compared to their real-world paths. These results are shown in Figures 5.11 to 5.13. The initial position of the robot was on the x-axis and the starting position was calculated as the midpoint of the tracked markers which was located at coordinates (30,0) for all tasks. The yellow and green tracked markers are represented by the $\square$ and $\triangleright$ annotations, respectively. The order in which goal regions needed to be traversed are represented by the numbered circles. The dashed lines represent the simulated path of the validation controller and the solid lines represent the real-world paths followed by the robot. The paths in simulation or reality were calculated as the path followed by the midpoint between the tracked markers. For each validation controller, three real-world evaluations were performed. The simulated paths presented in these figures do not include any noise.

There were twelve independently evolved controllers using the pre-computed SNNs, of which only the best three are shown in this work. Three trials were excluded as the robot moved out of the bounds of the operating surface. Another three trials were excluded as the robot rolled during transitions between cycles. One trial contained a failed cycle and two trials were left out due to poor transference to the real-world robot. The high number of trials required to find viable controllers in-

dicated that there was a reality-gap problem. The differences in behaviours observed in simulation and reality were not significant enough that they made it impossible to develop viable controllers.

Evaluated cycles exhibited many types of locomotion modes, such as lateral strafing, side-winding, helical movements, rolling, flapping and many others. Certain behaviours, such as flapping, were observed to be unable to transfer well to reality whereas other behaviours such as side-winding showed better transference to reality. Transitions between different cycles could also cause stability and accuracy problems. The same sequence of commands could generate varying behaviours on the real-world robot due to a lack of repeatability in the movements and this could affect the accuracy of the results.

The controller search space was large and often contained regions where small changes to a controller could result in large changes in behaviour. For each trial, the number of cycles used, percentage of goal regions reached, average displacement between the final positions in simulation and reality and lastly the number of controller evolution generations of the validation controllers are given in Table 5.2.

Inaccuracies between the simulator and real-world could be due to many factors, such as path divergence over time due to an accumulation of errors. The simulator was simplified and did not predict changes in position due to transitions between cycles. The tracked marker positions in simulation could also drift too far or too close to each other during controller evaluations, resulting in unrealistic behaviours in simulation.

The first cycle in trial 1 (Figure 5.11) moved the robot into the initial goal region. The second cycle reversed the robot's direction towards the origin of the working space. The slight shift eastwards around the origin was the result of the robot's shift in position due to switching to the third cycle. The third cycle was repeated three times and showed poor transference to reality which was possibly due to the lack of training data patterns near that region of the cycle search space. The last distinct cycle was repeated three times and brought the robot towards the last goal region.

Table 5.2: Trial run details

Trial number	**Cycles**	**Goal regions reached by real-world runs**	**Final displacement error**	**Controller generations**
Trial 1	8	89%	21cm	200
Trial 2	6	78%	16cm	100
Trial 3	8	100%	53cm	100

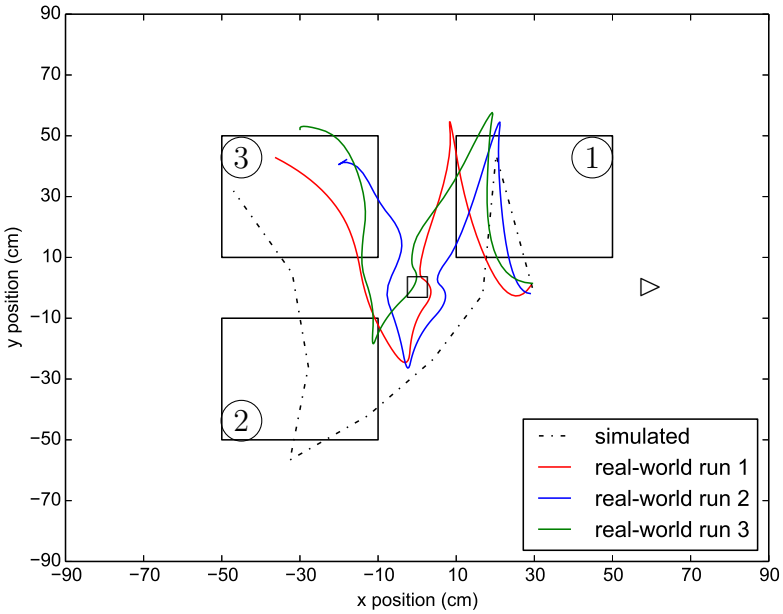


Figure 5.11: Experimental run, trial 1

The first cycle in trial 2 (Figure 5.12) moved the robot into the initial goal region and overshot towards the right when compared to the predicted path. The next two cycles moved the robot towards, and mostly into the second goal region. Only one of the real-world runs missed this goal region. The fourth cycle was supposed to move the robot westward. However, it resulted in the real-world robot holding the

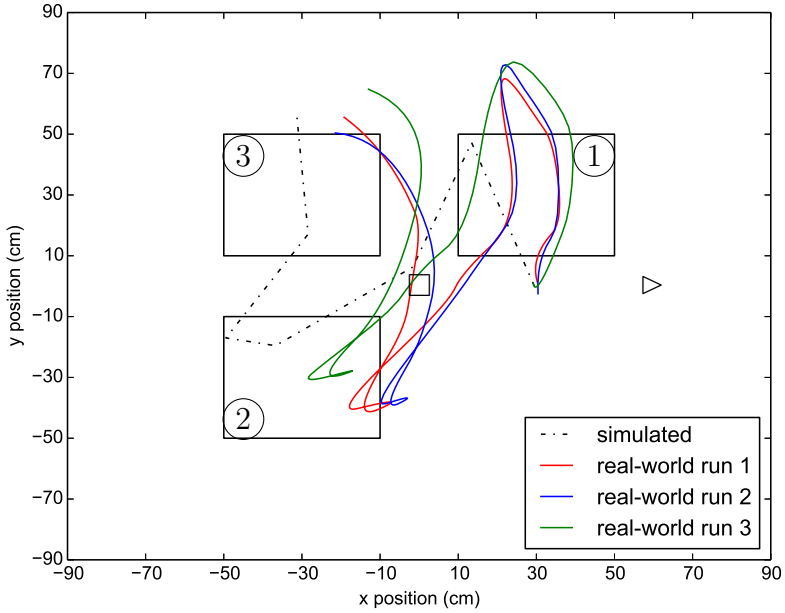


Figure 5.12: Experimental run, trial 2

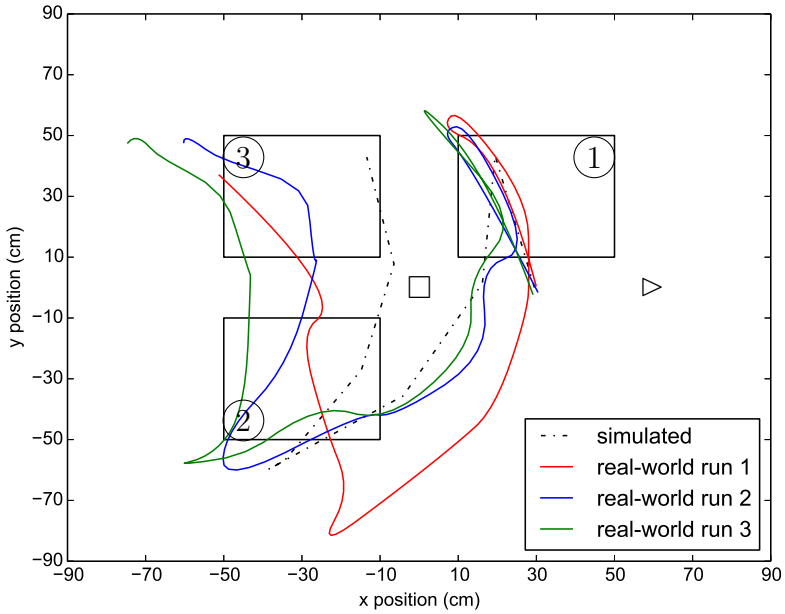


Figure 5.13: Experimental run, trial 3

same position. The last two cycles formed a curved path towards the third goal region which closely matched the predicted behaviour.

The first cycle in trial 3 (Figure 5.13) moved the robot over the first goal region and overshot it slightly towards the left. The next three identical cycles brought the robot towards and just below the second goal region. The fifth cycle moved the robot slightly closer towards the second goal region and the last three cycles moved the robot over the second and third goal regions.

5.5.3 Discussion

From the results shown in Section 5.5.1, it was observed that changes in position of the yellow marker could be modelled more accurately than those of the green marker. This could be due to the increased friction caused by the added weight of the Arduino and the tether on one end of the robot (Figure 5.4). The increased friction could cause the robot to exhibit less slippage. An environment or robot with different frictional properties could possibly yield better results.

The trajectory planning experiments (Section 5.5.2) demonstrated the viability of using SNNs to simulate snake robot behaviours and evolve effective trajectory planning controllers. Many independent trial runs of the ER process were required in order to identify effective controllers. The development of a simulator without inconsistencies between behaviours observed in simulation and reality was shown to be difficult due to the complexity of the behavioural search space. Future work may be able to identify ways of reducing the reality-gaps without too many real-world evaluations. Independent trial runs of the ER approach using the pre-computed SNNs resulted in controllers with very different behaviours, and certain behaviours transferred well to reality and others transferred poorly. Once the viability of using SNNs to simulate snake-like robot behaviours had been demonstrated, the experiments in the following section were used to determine the viability of concurrently developing SNNs and controllers by using the BNS approach.

5.6 BNS Prototype Experiments

The BNS approach was performed on the real-world snake-like robot (Section 5.3.1). Experiments proceeded until the robot was visually perceived to have traversed all the goal points or controller evolution had converged towards a consistent wrong solution. Each of the trajectory planning tasks was completed using the BNS approach over three trials.

Controllers consisted of encoded parameters for the variable terms in equations (5.1) and (5.2). The controller evolution settings are given in Table 5.3. Optimal parameter values were determined experimentally, based on the parameter comparison experiments described in Section 5.8.

Population Size	400
Initialization	Random from a uniform distribution
Selection	Tournament (size 200)
Cross-over Method	Uniform
Mutation Probability	10%
Mutation Method	Random Component Perturbation

Table 5.3: Parameters for controller evolution

An initial population of 400 controllers was generated randomly from a uniform distribution. Each controller consisted of ten unique cycles and each cycle could be repeated four times. During uniform cross-over operations, two parents were selected by using tournament selection (using a tournament size of 50%). Child controllers consisted of approximately 50% genetic material from one parent and 50% from the other. The probability that controller parameter values were mutated by a random amount was 10%. Elitism was used, with the best performing controller for each generation carrying over to the next.

An additional parameter which was specific to the BNS approach was the Real-

world Selection parameter. At the end of every real-world controller evaluation, a new controller was selected for the next real-world evaluation. This new controller was selected from the population of controllers by using tournament selection and the tournament size was specified by the Real-world Selection parameter. A Real-world Selection Size of 70% of the controller population was chosen for the experimental work. The following section presents the results of the prototype experiments performed on the real-world snake robot.

5.7 BNS Prototype Results

The real-world snake prototype experiments were able to demonstrate that the BNS approach is indeed viable on a complex, snake-like robot. The developed SNNs made sufficiently accurate predictions and controllers were successfully evolved to exhibit the required behaviours on a real-world robot. Controllers were developed for two types of trajectory planning tasks. For Task 1 (Figure 5.14), the robot was initially situated between two goal regions. Task 2 (Figure 5.15) had three goal regions positioned around the origin of the operating surface and were spaced 20cm apart. The robot was initially placed along the x-axis and the starting position was the midpoint of the tracked markers, located at coordinates (30,0) for both tasks. The yellow and green tracked markers are represented by the $\square$ and $\triangleright$ annotations, respectively. The order in which goal regions needed to be traversed are represented by the numbered circles.

The total run-times and number of real-world evaluations for each trial is given in Table 5.4. The run-times do vary considerably and are highly dependent on the developed behaviours. The Task 2 run-times were more than double the Task 1 run-times which can be ascribed to the greater complexity of Task 2 which required the exploration of more robot behaviours. The addition of an extra goal region did not appear to increase the complexity of the task linearly, based on the run-times and the number of real-world evaluations. It was observed in Task 2 that the

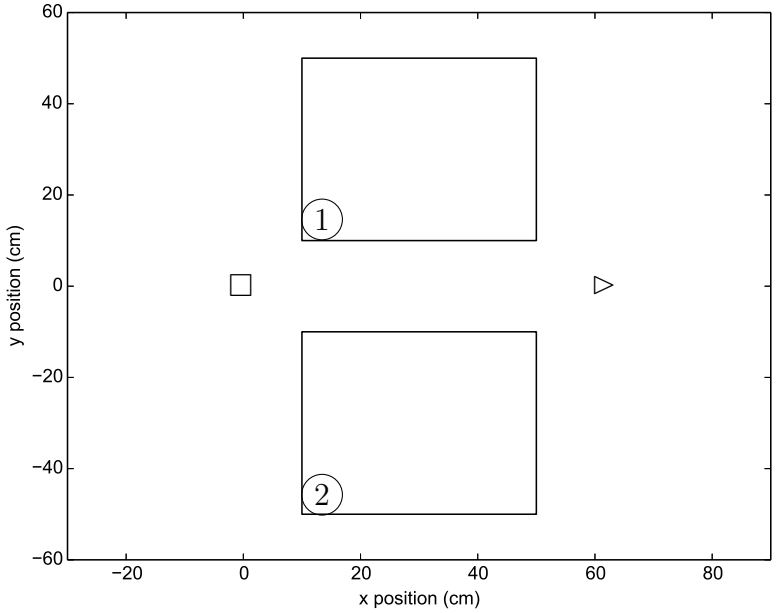


Figure 5.14: Task 1

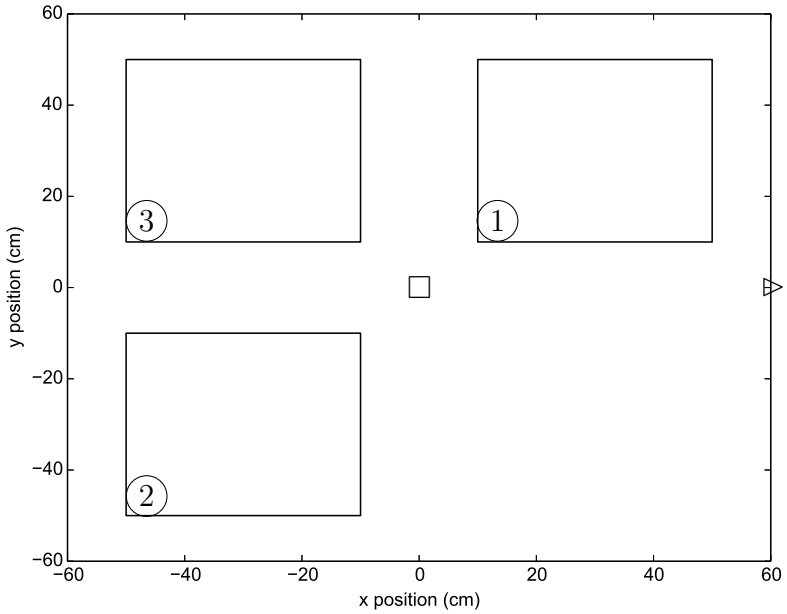


Figure 5.15: Task 2

BNS approach could find solutions that allowed the robot to traverse the first two goal regions with run-times similar to Task 1, but the third goal region increased the difficulty more than expected. In each figure (Figures 5.16 to 5.21), the solid lines represent the average real-world paths of the midpoint of the tracked markers followed by the evaluated controller. Each controller was evaluated three times on the real-world robot in order to gauge the repeatability of results. The dotted line represents the developed simulator's simulated path followed by the best identified solution of the ER process.

The real-world paths demonstrated in Figure 5.16 closely matched the simulator's perceived path with little difference between the real-world runs. The robot moved upwards which was followed by a downward motion forming an arc as one end of the robot moved further than the other. The controller consisted of two distinct cycles, one cycle moved the robot upward and the next cycle was repeated twice and moved the robot downwards. The simulator accurately predicted the real-world behaviours and the goal task was successfully completed by the developed controller when transferred to the real-world robot.

Goal task	**Trial number**	**Experiment run-times**	**Real-world evaluations**
Task 1	Trial 1	68 minutes	19
	Trial 2	96 minutes	23
	Trial 3	102 minutes	31
Task 2	Trial 1	200 minutes	42
	Trial 2	245 minutes	53
	Trial 3	210 minutes	46

Table 5.4: Prototype run-times

The solution demonstrated in Figure 5.17 contained the largest number of cycles

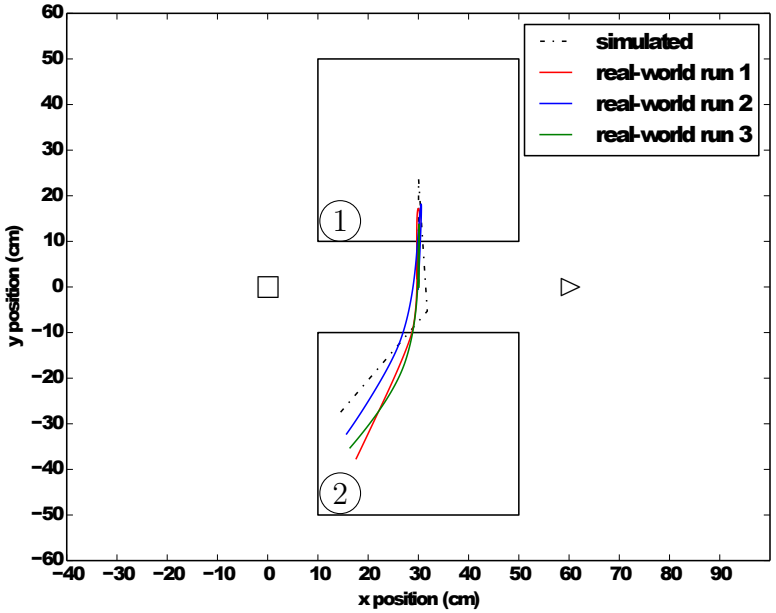


Figure 5.16: Task 1, first trial run at the 19th real-world evaluation

out of the three trials for Task 1. The robot used two cycles to move upwards into the first goal region and closely matched the predicted paths. The next cycle was perceived by the simulator to move the robot to the left, but in reality the robot remained in place. The fourth cycle was supposed to move the robot downwards whilst within the goal region, however, the robot was observed to move upward which was mainly due to the shift in position when transitioning between the third and fourth cycles. The fifth cycle was repeated twice and was responsible for moving the robot out of the first goal region and into the second goal region.

The third trial run for Task 1 (Figure 5.18) was able to accomplish the task and the predictions transferred reasonably well to reality. The first cycle brought the robot into the first goal region which corresponded well to the predicted path, however, the transition to the next cycle was observed to result in a minor shift towards the right in the real-world runs. The second cycle was repeated twice with the predicted distance covered by the robot being slightly underestimated by the

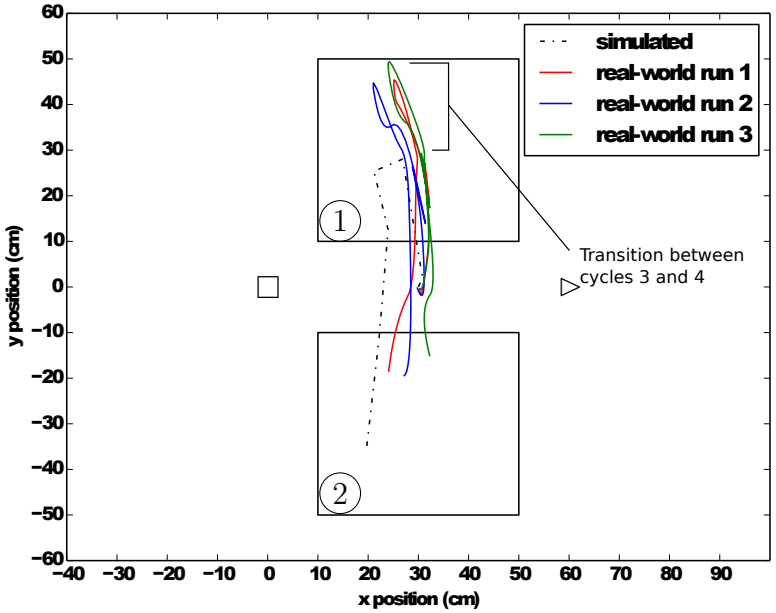


Figure 5.17: Task 1, second trial run at the 23rd real-world evaluation

simulator, but it still accurately simulated the general behaviour. The simulated path appeared to overshoot the second goal region which was due to the addition of noise. If the end of the last cycle missed the second goal region, then the goal would be reached by the second last cycle and if the second last cycle missed the second goal region, then the goal would be reached by the last cycle.

The first trial run of Task 2 is shown in Figure 5.19. The robot started along the x-axis below the top right goal region. The first cycle moved the robot barely into the first goal region which closely matched reality. The simulator estimated that the second cycle would end near the top left corner of the first goal region, but the real-world runs did not transfer this predicted behaviour. The transition from the second to the third cycle was observed to result in the robot slipping which explained differences between real-world runs. The next cycle was repeated three times and was predicted to bring the robot into the second goal region which closely matched the real-world behaviour. The next cycle shifted the robot in position and

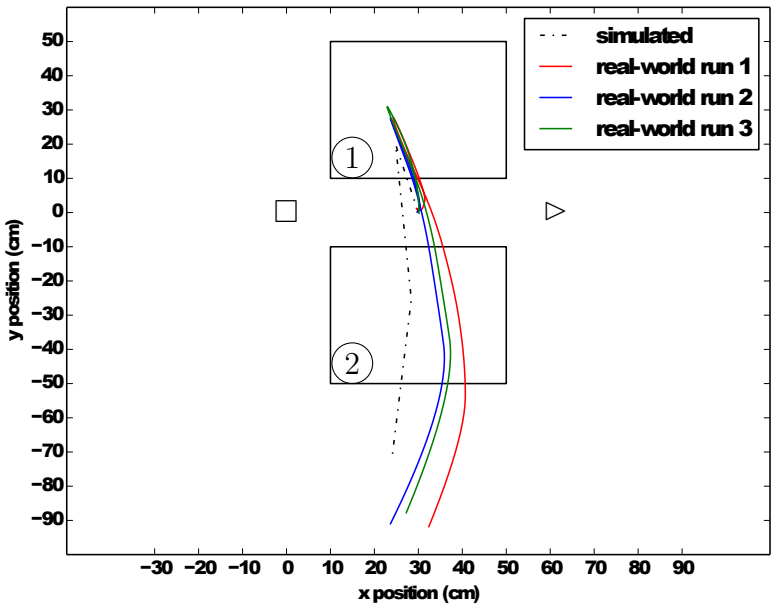


Figure 5.18: Task 1, third trial run at the 31st real-world evaluation

the last three cycles which were predicted to bring the robot out of the second goal region and into the final goal region accurately matched the real-world behaviour. The real-world runs often did not reach the second goal region due to the poor transference of the second cycle and the slippage observed between the second to third cycles.

Figure 5.20 demonstrates the second trial of Task 2 which appeared to mostly reach the first two goal regions, but transferred poorly for the last goal region. The trial run converged towards the behaviour shown in Figure 5.20 and did not improve much after the fifty-third real-world evaluation even though the trial was run up to the eightieth real-world evaluation. The first cycle moved the robot upward and closely matched the simulated behaviour, but there was slipping when transitioning to the second cycle. The simulator over-predicted the distance that was covered by the second cycle. The second cycle was repeated twice and was supposed to carry the robot into the second goal region, however, most of the real-world runs moved

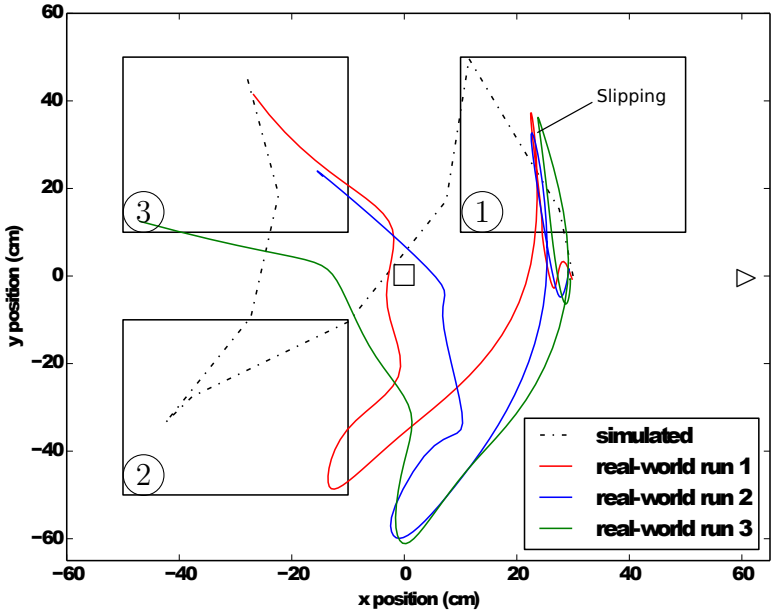


Figure 5.19: Task 2, first trial run at the 42nd real-world evaluation

the robot into the second goal region at the end of the third cycle. The fourth cycle was predicted to move the robot slightly to the left within the goal region, however, in the real-world very little change in the robot's position resulted. The last cycle was repeated four times and was supposed to allow the robot to reach the final goal region but the curvature of the real-world paths was not as acute as simulated. The lack of curvature may have been because the tracked markers drifted unrealistically too close towards each other during simulator evaluations.

In Figure 5.21, the robot was able to perform the goal task even though there was a high degree of slipping. The first cycle transferred well to reality and brought the robot barely into the first goal region and the next cycle was repeated twice and brought the robot barely within the second goal region. The fourth cycle moved the robot leftwards and further into the second goal region. The last cycle was repeated three times and was observed to cause a high degree of slipping, but moved the robot into the third goal region.

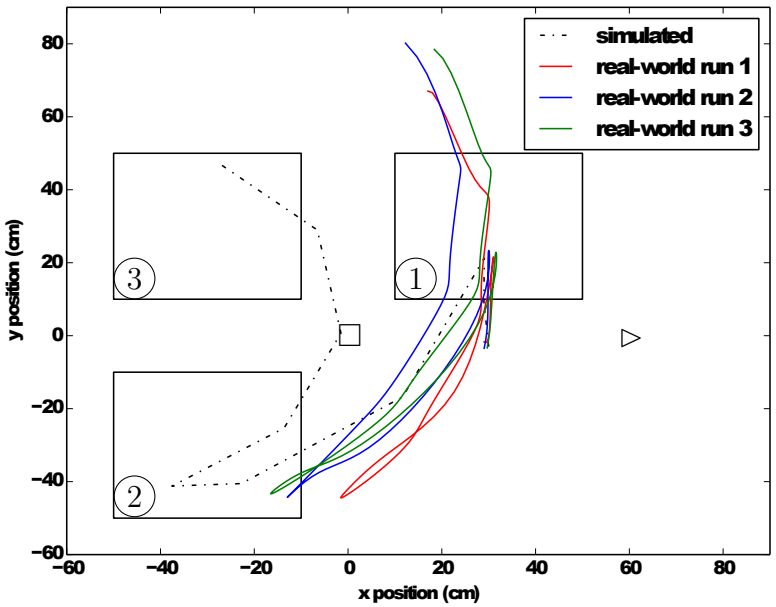


Figure 5.20: Task 2, second trial run at the 53rd real-world evaluation

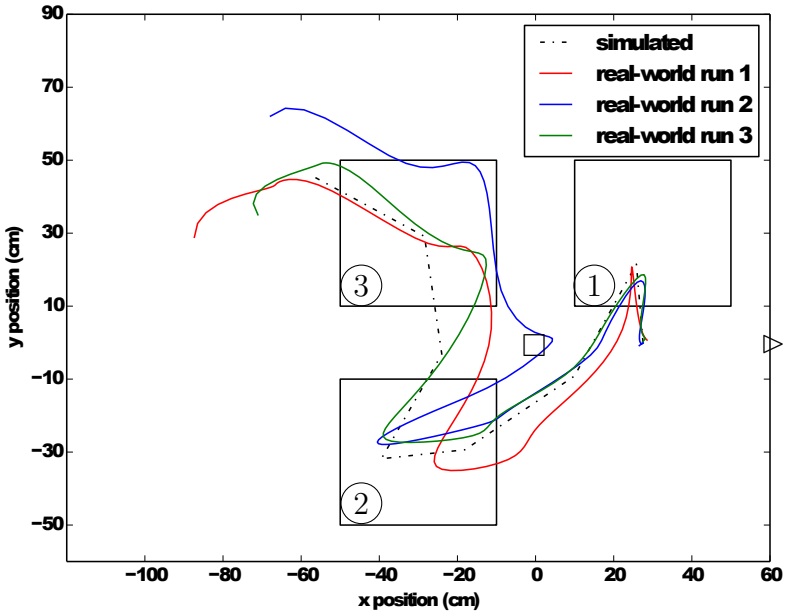


Figure 5.21: Task 2, third trial run at the 46th real-world evaluation

Task 1 required relatively few real-world evaluations and the developed simulator made sufficiently accurate predictions of reality. Task 2 was more complex and required nearly double the number of real-world evaluations compared to Task 1. For Task 2, the most difficult part of the task was moving the robot from the second to the third goal region. The orientation of the robot at the second goal region could vary from one solution to the next, requiring the exploration of a much larger set of behaviours to account for the changes in orientations.

The BNS approach does not guarantee that a successful solution can be found for every independent run. Controllers could converge on a solution that did not transfer well to reality, such as the second trial of Task 2 where controllers converged on a solution that was unable to reach the third goal task even though a large number of real-world evaluations was conducted. The BNS approach could fail to find successful controllers due to many reasons, such as poorly transferred, constantly slipping or unstable cycles. There could be a large change in the position of the robot when transitioning between cycles. The distance between the tracked markers may increase or decrease unrealistically during a controller's evaluation in simulation which could result in inaccuracies.

5.8 Parameter Comparisons

In order to investigate important factors that contribute to the success or failure of the BNS approach on the snake robot, the influence that various parameter settings had on success was investigated. The parameter settings tested are identical to the Khepera parameter comparison experiments (Section 4.5) and are listed in Table 4.3. Due to the stochastic nature of the experiments, every possible combination of the parameter settings was independently run for 30 trials for each task.

The investigation of such a large number of parameter combinations on a real-world robot would take infeasibly long due to the time required to run them. A pre-computed simulator was developed specifically for the snake robot and was used

as a substitute for the real-world robot, referred to as the static simulator (Section 5.4). Training data generated from the real-world robot would inevitably contain errors due to inconsistencies in motor function [Pretorius, 2010]. To accurately simulate reality, noise was thus added to the static simulator in order to realistically replicate noise that would be present in the real-world. The noise distribution was assumed to be Gaussian with mean zero because the real-world noise data had a mean close to zero and had a variance of 8cm for all ANNs of the simulator. The noise was based on the difference between the static simulator and real-world data. The variance was chosen to be higher than the observed training differences in order to improve the robustness of solutions. The static simulator with noise which acted as a substitute for the real-world robot will be referred to simply as the substitute real-world. Each experimental run was progressed until the one hundredth controller was evaluated in the substitute real-world.

The best fitness value during controller evolution was periodically recorded and the average final fitness of each parameter combination over the 30 trial runs was calculated. The success rate for a given parameter combination was the number of trial runs that successfully completed the goal task in the substitute real-world over the total number of trials (30). The success rates, along with the average fitness of controllers evaluated for each parameter combination were used to rank how well each combination performed overall.

The relationship between the diversity and various other properties, such as parameter settings, controller success rates and how well the developed simulator was able to predict the substitute real-world fitnesses, were also studied. The diversity D for a given controller population was calculated using equation (5.3).

The parameter settings for all controllers were normalized to ensure that there was equal weighting of importance. The normalized $A_{lateral}$, $A_{vertical}$, ω, α and cycle repeats of the j^{th} cycle for the i^{th} controller in the population of controllers are denoted by $A_{l_{ij}}$, $A_{v_{ij}}$, ω_{ij}, α_{ij} and C_{ij} respectively. The controller population size is denoted by n and the number of cycles needed to complete the given task in

simulation for the i^{th} controller is denoted by r_i. The average normalized $A_{lateral}$, $A_{vertical}$, ω, α and number of cycle repeats for the j^{th} cycle over all controllers is denoted by $\overline{A}_{l_j}$, $\overline{A}_{v_j}$, $\overline{\omega}_j$, $\overline{\alpha}_j$ and $\overline{C}_j$ respectively.

$$D = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{r_i}(|A_{l_{ij}} - \overline{A}_{l_j}| + |A_{v_{ij}} - \overline{A}_{v_j}| + |\omega_{ij} - \overline{\omega}_j| + |\alpha_{ij} - \overline{\alpha}_j| + |C_{ij} - \overline{C}_j|) \tag{5.3}$$

The fitness error was calculated by computing the absolute difference between a controller's fitness in simulation versus the substitute real-world. At the end of each substitute real-world evaluation, data was collected regarding the controller population diversity, along with the fitness error and success of the fittest controller. The diversity provided an indication of the balance between exploration and exploitation of the controller evolution process. A high diversity indicates a large level of exploration where controllers continually explore the search space in order to find new solutions. A low diversity indicates a high level of exploitation where controllers try to improve upon existing solutions.

5.9 Parameter Comparison Results

The results of the parameter combination experiments described in the previous section are now presented. The experiments were conducted completely in simulation, based on the pre-computed simulator developed in Section 5.4. Section 5.9.1 discusses the results that relate to the success rates of the parameter combination experiments and Section 5.9.2 relates success to the diversity. Section 5.9.3 discusses results related to the best and worst performing parameter combinations.

5.9.1 Success Rate of the BNS Approach

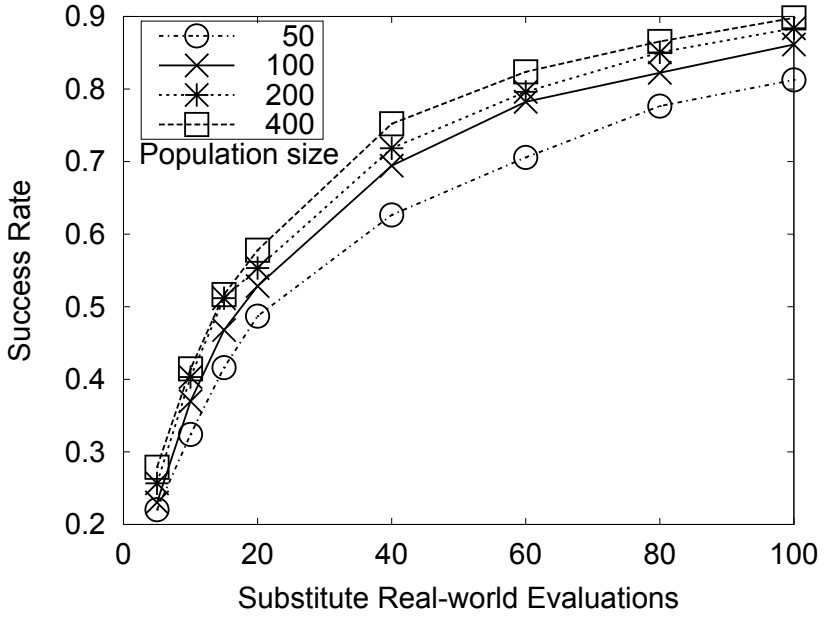
The success rate versus the number of real-world evaluations for each of the parameter values listed in Table 4.3 was determined over all other parameters. These

success rates were used to determine which parameters were most influential.

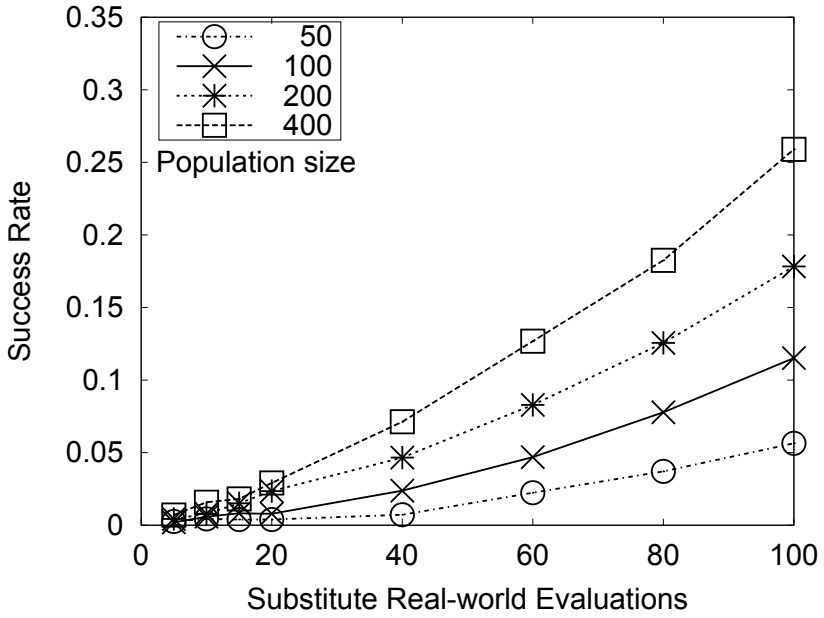
The success rates of the various controller population sizes, over all experiments, are shown in Figure 5.22. The results presented here demonstrate that the most influential parameter tested for influencing success rates was the controller population size. It was observed that larger controller population sizes generally performed better overall. The largest difference between the population size success rates was approximately 12.6% at the fortieth substitute real-world evaluation for Task 1 and 20.3% at the one hundredth real-world evaluation for Task 2. The success rates appeared to almost reach saturation levels between 80%-90% for Task 1 and the success rates for Task 2 were seen to increase linearly approximately between the twentieth and one hundredth substitute real-world evaluations.

The success rates of the various controller mutation rates over all experiments is shown in Figure 5.23. The success rates demonstrated that the controller mutation rate was the second most influential parameter tested. The widest gap between success rates for Task 1 occurred around the twentieth substitute real-world evaluation which had the largest difference between success rates at approximately 14%. The largest difference between the mutation success rates for the one hundredth substitute real-world evaluation was 1.2% for Task 1 and 14.2% for Task 2. The small gap in mutation success rates for Task 1 could be because the success rates reached a saturation point which was approximately between 80-90%. For Task 1 the 10% mutation rate performed the best over the lifetime of the experiments, followed by a 30% mutation rate. For Task 2 the best mutation rate was also 10% followed by 30% and the last three, worst mutation rates performed similarly well (within 2% of each other).

The success rates over time for the controller and real-world tournament selection sizes are given in Figures 5.24 and 5.25, respectively. The results showed that the controller and real-world tournament selection sizes performed almost equally well with no more than a 3% difference between the success rates for both tasks. The largest difference between the controller tournament selection size success rates were

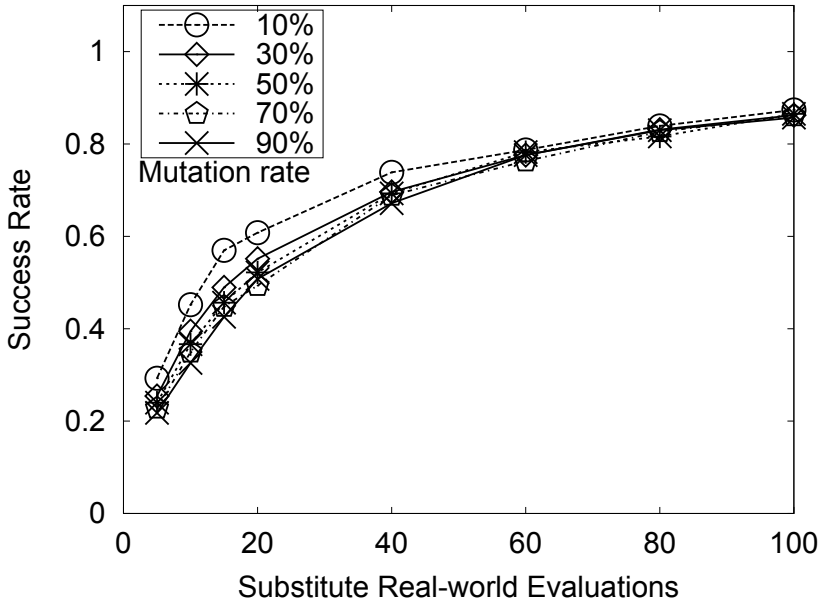


(a) Task 1

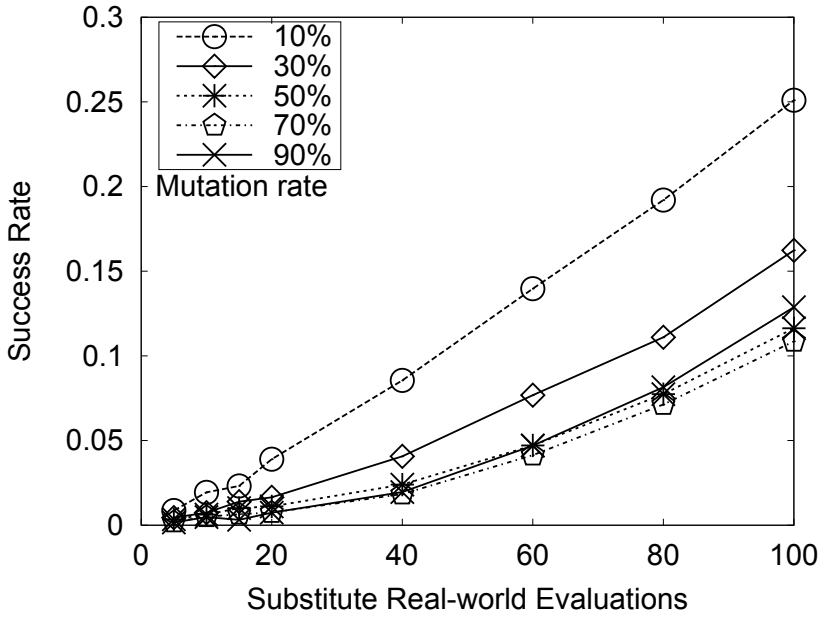


(b) Task 2

Figure 5.22: Success rate versus number substitute real-world evaluations for the tested controller population sizes



(a) Task 1



(b) Task 2

Figure 5.23: Success rate versus number substitute real-world evaluations for the tested controller mutation rates

3% for Task 1 and 2.3% for Task 2. The largest difference between the real-world tournament selection size success rates were 2.6% for Task 1 and 2.5% for Task 2.

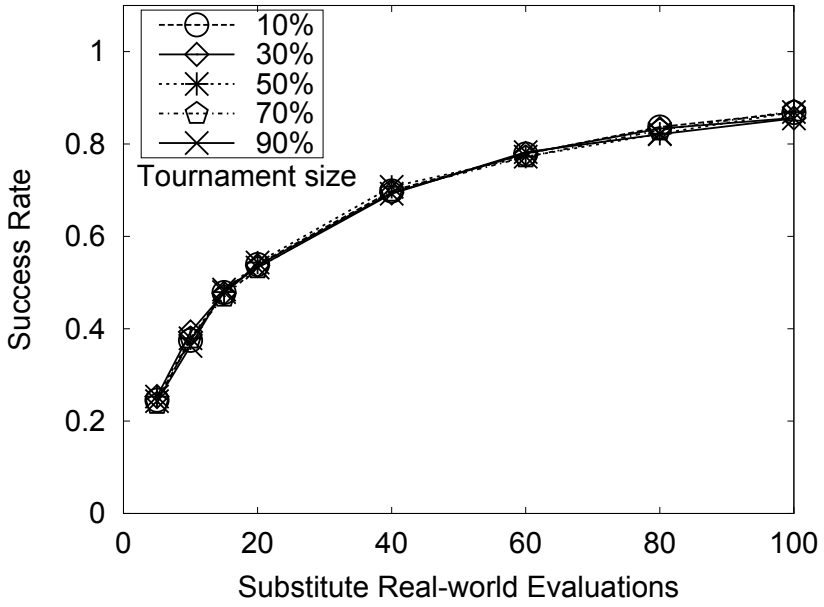
5.9.2 Diversity

In order to determine how accurately the simulators performed in relation to diversity, the fitness error was measured. The average fitness error versus population diversity for each parameter combination after the one hundredth substitute real-world evaluation can be seen in Figure 5.26. The plotted parameter combinations were colour-coded according to the controller population size. A low fitness error indicated that the developed simulator's predicted fitness of the final controller closely matched its substitute real-world fitness. The success rate versus diversity for each parameter combination is shown in Figure 5.27 which is colour-coded according to the controller mutation rate.

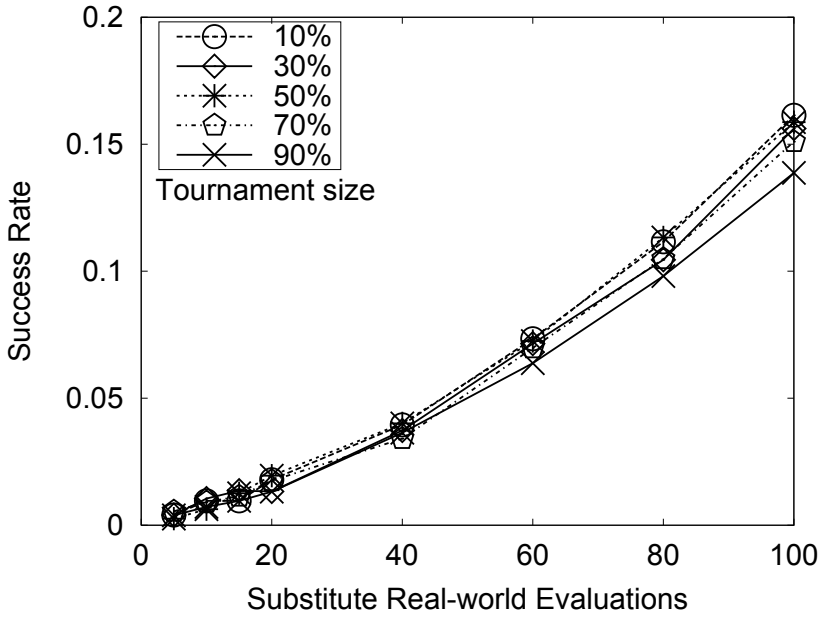
Parameter combinations with a greater population size were observed to generally have a lower corresponding fitness error for both tasks (Figure 5.26). Controllers selected for real-world evaluations in the larger populations may have been more likely to be closer to the fittest controller in the population which could account for the lower fitness errors. A greater population size may have improved the retention of high fitness solutions and reduced the chance that good solutions could be lost due to a changing simulator which would result in more stable controller populations.

The fitness error versus diversity for Task 1 (Figure 5.26a) had two main clusters centred around a diversity of 2 or 3.4. For each cluster, a greater population size tended to have a larger diversity which could be due to larger population sizes containing a wider variety of solutions and perhaps converging slower than smaller population sizes.

The fitness error versus diversity for Task 2 (Figure 5.26b) was not observed to form two separate clusters, as was seen in Task 1, however, the parameter combinations appeared to cluster together based on their population sizes. As in Task

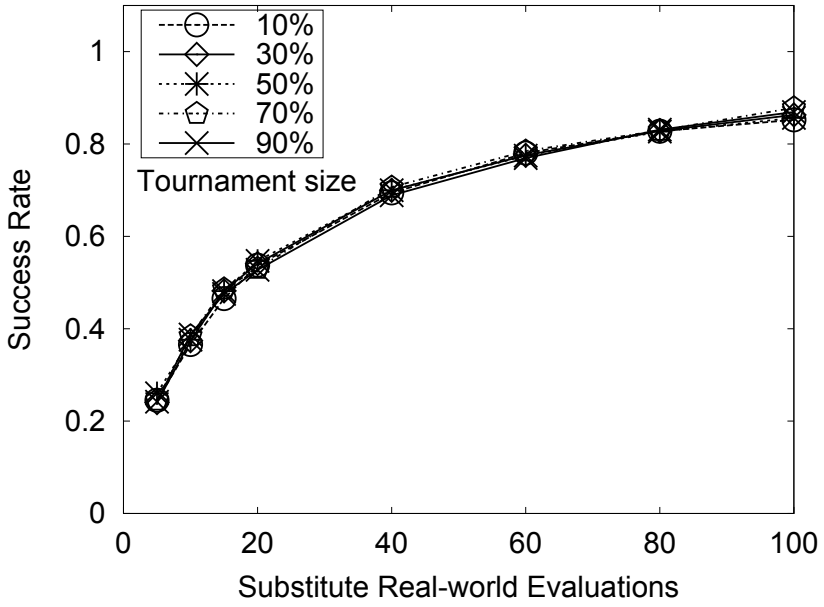


(a) Task 1

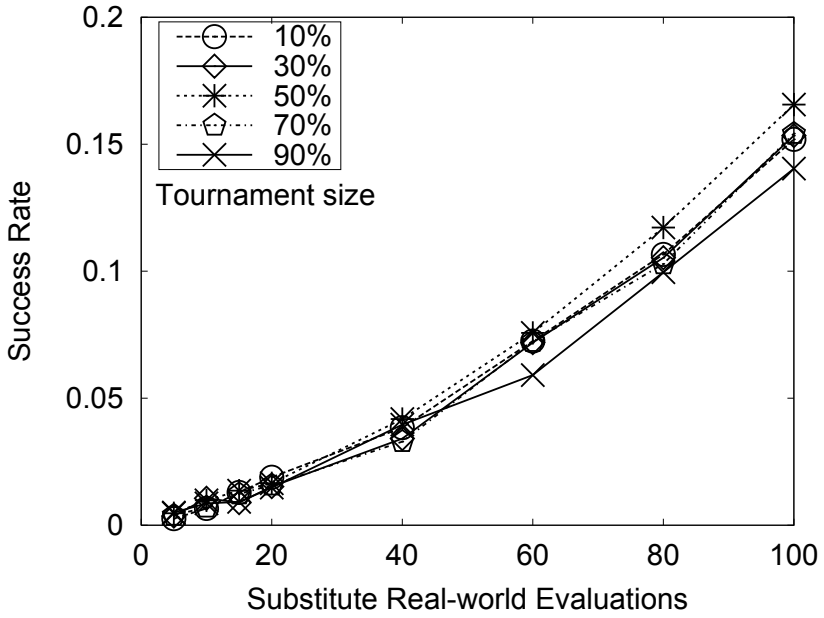


(b) Task 2

Figure 5.24: Success rate versus number substitute real-world evaluations for the tested controller evolution tournament sizes

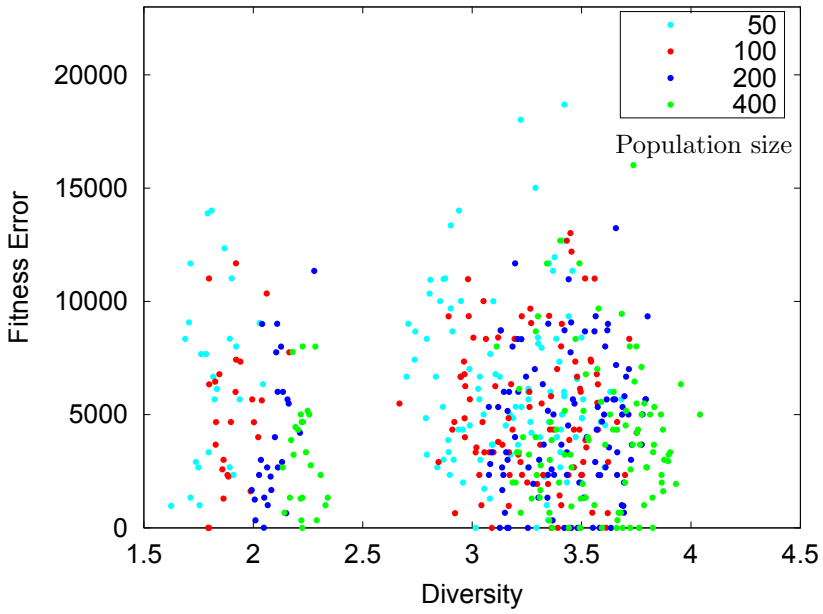


(a) Task 1

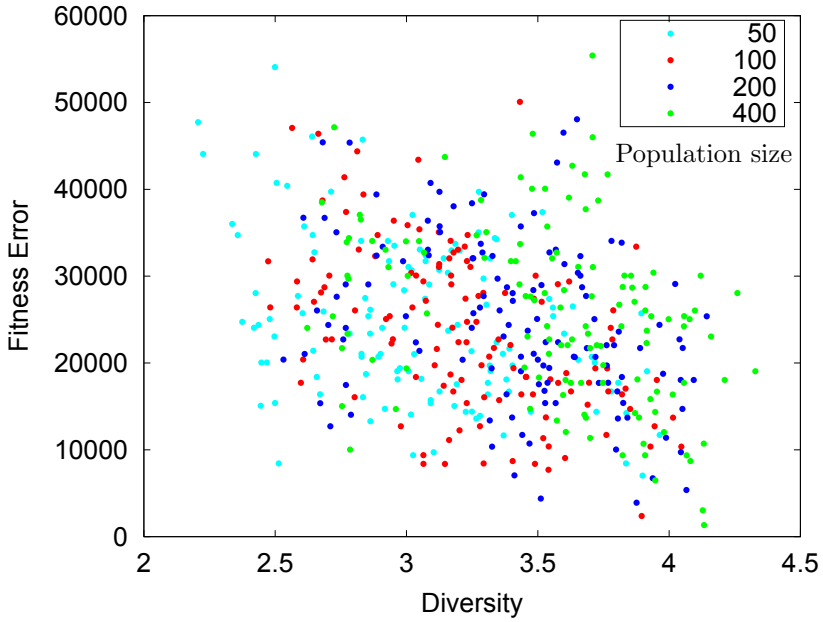


(b) Task 2

Figure 5.25: Success rate versus number substitute real-world evaluations for the tested real-world tournament sizes



(a) Task 1



(b) Task 2

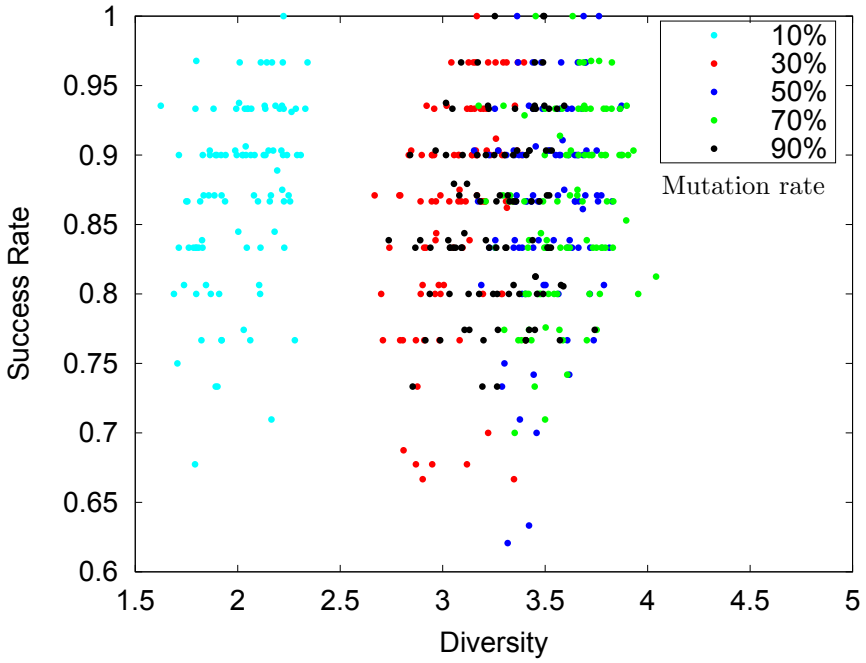
Figure 5.26: Average fitness error versus diversity after 100 substitute real-world evaluations grouped by population size

1, larger population sizes tended to have a greater diversity. The range of fitness errors was much larger for Task 2 which may be due to the greater complexity of Task 2 compared to Task 1. The fitness errors for poorly predicted behaviours could vary greatly for independent trials and the fitness errors for poorly predicted trial runs could unfairly distort the average fitnesses, resulting in a wider range of fitness errors.

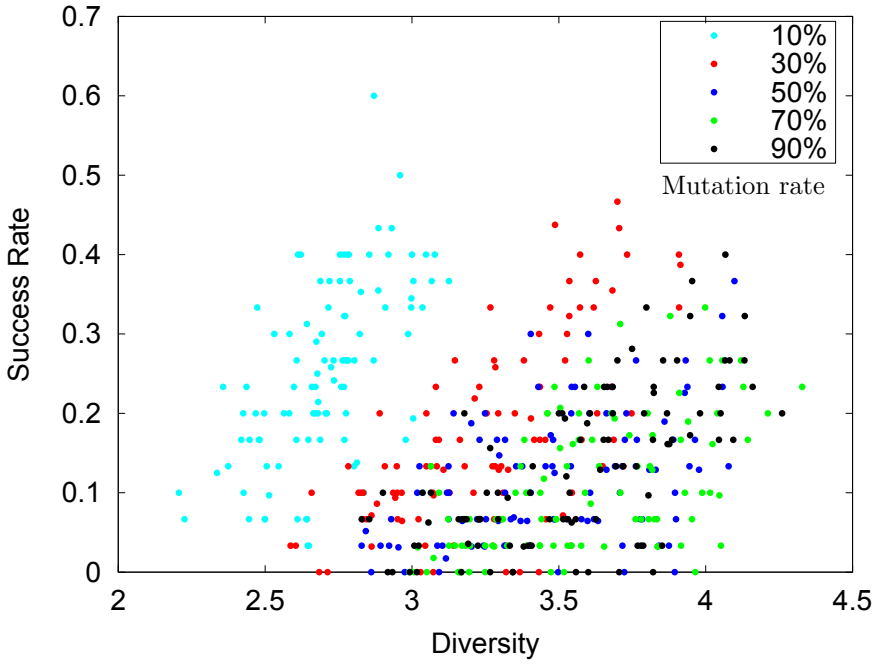
The Task 1 and 2 success rates versus diversity for the various mutation rate groupings are shown in Figure 5.27. For both tasks, the mutation rate for all the parameter combinations in the lower cluster consisted of a 10% mutation rate and the higher diversity cluster contained mutation rates of 30% and above. The optimum success rates were achieved for each mutation rate grouping through a balance between the exploitation and exploration of controller populations. The parameter combinations with a lower population size tended to have a lower success rate and a lower diversity which was demonstrated when Figure 5.27 was redrawn (Figure 5.28) for only those parameter combinations with a population size of 400.

A lower diversity may have resulted in a lower success rate due to the exploitation of sub-optimal solutions and a high diversity may have resulted in the exploration of newer behaviours before existing solutions could be refined. The optimum balance between exploration and exploitation for achieving the best success rates for each mutation rate grouping was clearly skewed towards greater exploration (higher diversity), but the success rates tended to drop for the highest diversities. This demonstrated that the exploitation of sub-optimal solutions was the greatest problem related to the BNS approach. A larger population size helped to increase the degree of exploration and reduced the likelihood of the evolution process converging on sub-optimal solutions. Low mutation rates were more likely to have a high success rate which may have been due to it facilitating the exploitation of the diverse set of solutions guaranteed by the large population size.

For Tasks 1 and 2, the success rate versus diversity results (Figure 5.27) did not demonstrate the same trends observed in the Khepera success rate versus diversity



(a) Task 1



(b) Task 2

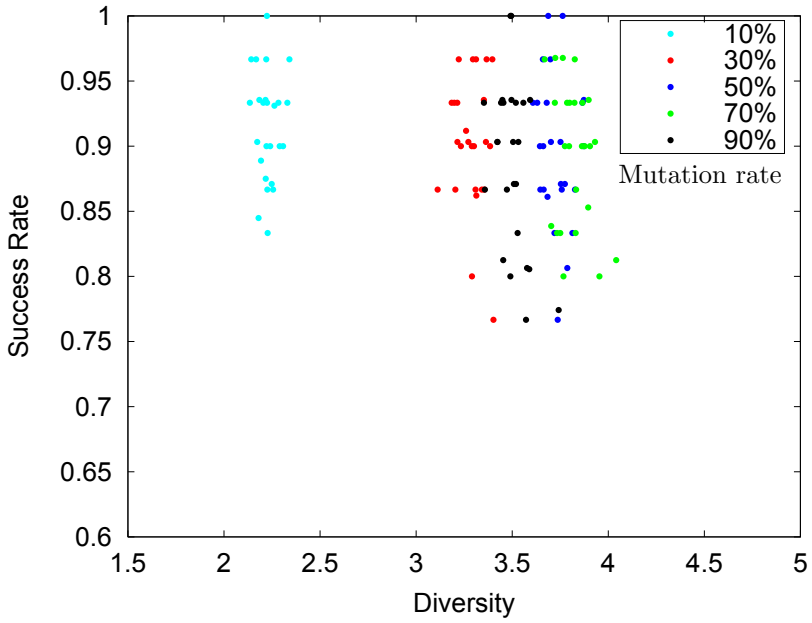
Figure 5.27: Success rate versus diversity after 100 substitute real-world evaluations grouped by mutation rate

results (Figure 4.12). The trend observed for Khepera experiments (Section 4.6.2) demonstrated that a very high or low diversity resulted in a poor success rate and a medium-low optimum diversity range performed the best with few poor performers around this optimum region. The entire diversity spectrum for the snake robot contained many poorly performing solutions. There were good and bad solutions within the optimum diversity range for the snake robot. This was due to the greater influence that the controller population sizes had on the success rate for the snake robot. As previously explained, a lower population size significantly decreased the likelihood of developing successful solutions. Figure 5.27 was redrawn (Figure 5.28) for only those parameter combinations with a population size of 400 which resulted in the majority of the badly performing solutions in the optimum diversity range disappearing. The controller search space for the snake robot was not as smooth as the Khepera's because of the difference in complexity in the behaviours. For the snake robot, changes to the simulator or controllers may be more likely to distort known good solutions and this was counteracted by a larger population size.

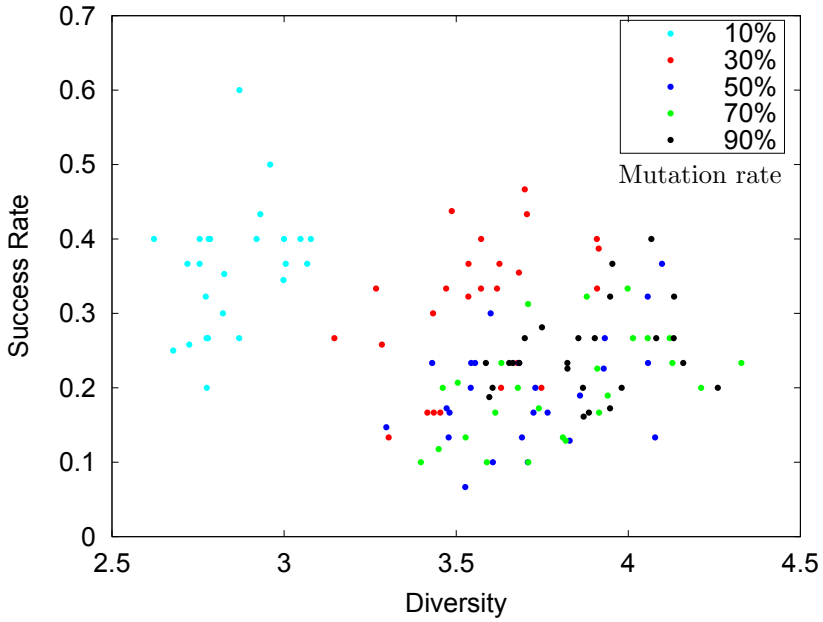
5.9.3 Best and Worst Performing Parameter Combinations

The best and worst performing parameter combinations were compared with each other in order to identify influential factors. The diversity versus the number of substitute real-world evaluations for the top 10 best and worst performing parameter combinations for Tasks 1 and 2 are shown in Figures 5.29 and 5.30, respectively. There was little point in analysing how the diversity behaved before the fifth substitute real-world evaluation, as controller evaluations were almost random. For this reason, the figures do not present the diversity before the fifth substitute real-world evaluation.

The top 10 best performing parameter combinations had a decreasing diversity over time for Task 1 and an increasing diversity for Task 2. For Task 1, the top performing diversities were possibly larger during the earlier stages of the ER process



(a) Task 1



(b) Task 2

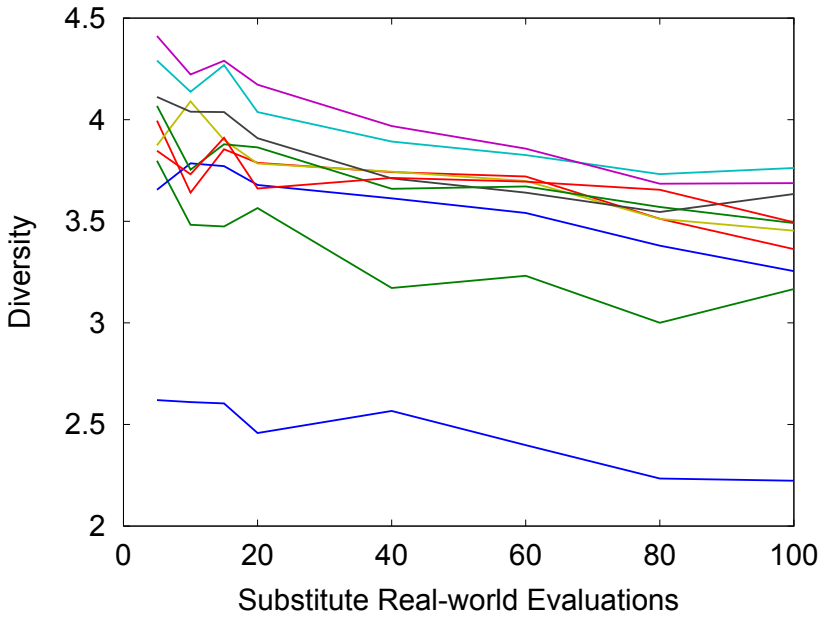
Figure 5.28: Success rate versus diversity after 100 substitute real-world evaluations for a population size of 400 and grouped by mutation rate

due to the exploration of the controller search space and the non-convergence of controllers due to large changes in the simulator during the early stages of the BNS approach. Task 1 was a simple task and controller populations were quickly able to converge towards particular solutions which could account for the decrease in the diversities over time. For Task 2, the diversity increase was possibly due to the greater complexity of the task where there was a larger search space for the controller evolution process to explore. The controller populations for Task 2 may converge slowly towards a full solution, but potentially converge towards partial solutions during the earlier stages of the BNS approach. Controller populations could converge on a solution that traversed two of the three goal regions, but the last goal region was more challenging to reach.

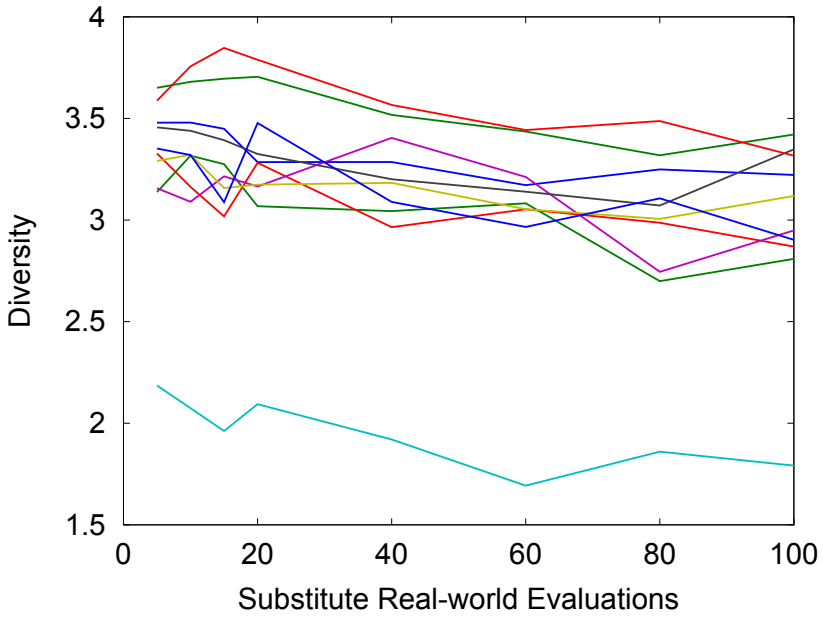
The worst performing diversities were fairly unstable and did not steadily increase or decrease over time (Figures 5.29b and 5.30b). For the worst performers, the continual increases and decreases in diversity throughout the lifetime of the process could indicate that controller evolution may be periodically losing good solutions. A steady decrease in the diversities may indicate that the controller populations converged towards solutions and a steady increase may indicate that the controller population contains an increasing number of possibly different solutions. This may require that the simulator needs to learn new behaviours that can further slow down the controller evolution process.

5.10 Conclusions

The difficulty in developing adequate simulators increases as the complexity of the robots, environments and tasks increase. SNNs provide a simple, automatic way for researchers to construct models of reality and avoid too many costly real-world evaluations needed for many empirical simulation techniques. This chapter has demonstrated that SNNs are indeed viable for evolving behaviours for complex snake-like robots during the ER process.

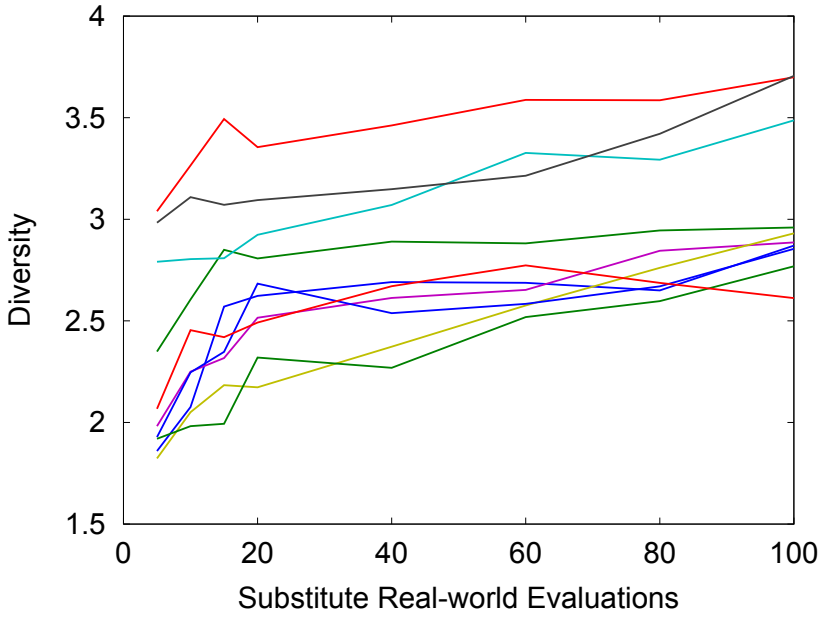


(a) Top 10 parameter combinations

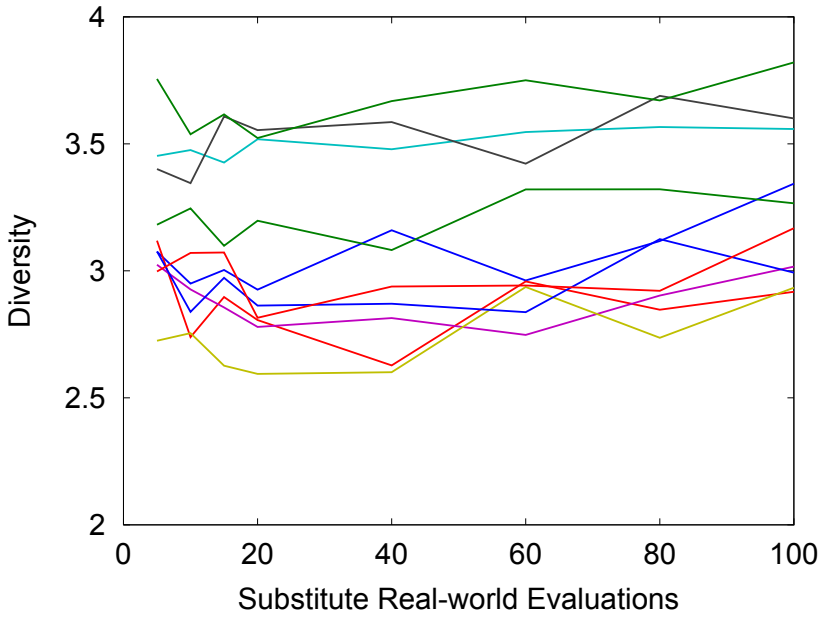


(b) Worst 10 parameter combinations

Figure 5.29: Diversity versus number of substitute real-world evaluations for Task 1



(a) Top 10 parameter combinations



(b) Worst 10 parameter combinations

Figure 5.30: Diversity versus number of substitute real-world evaluations for Task 2

The developed simulator was able to predict the general behaviour of the robot. It was also significant that behaviours could be evolved for a complex robot to navigate blindly in its environment with no feedback. This was noteworthy considering the many sources of errors and simplifications assumed by the simulator. Possible sources of errors included inaccuracies in motion tracking, inconsistencies in motor function, slippage on the operating surface and errors due to the simplifying assumptions of the simulator. Poor transference could also be due to insufficient training patterns around certain portions of the cycle search space. Much work needs to be done in devising better training data sampling strategies.

The complexities in the search space were significant, where small changes in cycle parameter settings could result in significant differences in behaviour. Cycles that did not transfer well to reality were noted and similar training data was generated which tended to improve transferability overall.

The collection of sufficient behavioural data was a time-consuming process and it was not feasible to get complete coverage of the search space. The BNS approach showed great potential in being able to develop solutions using less behavioural data compared with the precomputed approach, however, the BNS approach does not guarantee a successful solution and should be investigated further in order to develop improvements. SNNs require large amounts of behavioural data, but the development and tuning of a physics-based engine could become equally, if not more, time-consuming and would require specialised knowledge about the dynamics of the robotic system. SNNs and bi-directional approaches have been demonstrated in this research to be able to provide an alternative to traditional approaches.

Factors related to the success of the BNS approach were successfully identified. The real-world and controller evolution tournament selection sizes were the least influential parameters tested and no clearly discernible trends were identified. The controller mutation rate and population size parameters were observed to be the most influential parameters tested. A low controller mutation rate and a high controller population size was found to be the optimum choice. There needed to be a

balance between the exploration and exploitation of the controller search space. A low controller mutation rate allowed the evolution process to better exploit known good solutions, while a high population size allowed for a greater exploration of the controller search space.

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Introduction

This research developed and validated the BNS approach for the modelling of robot behaviours which was used to produce controllers for performing trajectory planning tasks using two robot morphologies. The outcomes for the research objectives are discussed in Section 6.2. The major significance of this research is detailed in Section 6.3 and the identified limitations are addressed in Section 6.4. The future potential of this research is discussed in Section 6.5 and finally an overall summary of this work is given in Section 6.6.

6.2 Overview of Results and Outcomes of Research Objectives

Research objectives were used to focus the scope of this study (Section 1.4). Each of the research objectives is now discussed.

Develop a new approach for the concurrent creation of SNNs and controllers in the ER process.

It was hypothesised that many of the disadvantages inherent in traditional simulator development could potentially be addressed by combining a bi-directional approach with an ANN-based simulator. Bi-directional approaches have been shown to reduce the number of real-world evaluations required to develop effective simulators whilst automating much of the simulator tuning process. SNNs have also been shown to reduce the amount of specialised knowledge required for the development of effective simulators. This research involved the creation and validation of a novel process for the concurrent development of SNNs and controllers in the ER process, called the Bootstrapped Neuro-Simulation approach. Potential advantages of such an approach may be a reduction in the number of real-world evaluations, although this was not explicitly investigated in this study. Changes to the robot's morphology and/or environment are automatically accounted for by this approach because a new simulator is developed for every experiment. The viability of this new approach was successfully validated through the development of real-world prototypes.

Identify and test the various factors pertinent to the success of the proposed approach.

A thorough study of the BNS approach was conducted in order to identify the conditions required to optimize success. The parameter comparison experiments identified factors pertinent to success and observed that certain parameters are more important than others for the specific case studies conducted in this work. The identification of significant parameter settings is a major contribution of this research (Sections 4.6.1 and 5.9.1). It was observed that the BNS approach required a balance between exploring new solutions in the search space, in order to find improvements, and the exploitation of existing solutions that may converge towards

the desired behaviours (Sections 4.6.2 and 5.9.2).

The identified parameter values and balance between the exploitation and exploration of the search space could potentially be applied to other robots and tasks not investigated in this research which is an important contribution for future research. In addition to these experiments, weaknesses of the BNS approach were identified that could be eliminated in future work.

Consider the scalability and generality of the approach by assessing different robot morphologies.

The Khepera robot was chosen due to its simple morphology and the initial viability of this research had yet to be demonstrated. The BNS approach was shown to successfully scale by choosing a second experimental robot of greater complexity and developing effective controllers. Real-world evaluations for the snake robot took much longer than that of the Khepera robot due to slower evaluation times. The potential generality of the developed approach was demonstrated by the success of this research to handle two very different robots morphologies. The snake robot has a high number of degrees of freedom and is relatively difficult to control compared to the Khepera robot. This research demonstrated that effective controllers could be developed for a snake-like robot which may indicate that the BNS approach could generalize well to many different types of robot morphologies.

Compare the effectiveness of the new approach for tasks of varying levels of complexity.

The Khepera (Chapter 4) and snake (Chapter 5) robot experiments consisted of three and two trajectory planning tasks, respectively. The complexity of trajectory planning tasks was varied in order to judge how well the BNS approach scaled to more complex tasks. It was observed that more difficult tasks required more

data and increased the number of real-world evaluations. The number of real-world evaluations required for evolving successful solutions was observed to be inconsistent. Greater task complexity appeared to more than linearly increase the amount of empirical data required for completing the given task. The BNS approach was, however, successfully able to develop viable controllers for the complex tasks.

Consider the limitations of the approach and identify future potential.

Previous research has shown that pre-computed SNNs are trained from the behavioural data of randomly generated controllers which allows the simulator to predict behaviours required for many different tasks. The BNS approach proposed in this research developed specialised SNNs that focused on predicting the behaviours of targeted controllers. Concurrently developed SNNs are therefore at a disadvantage to predict behaviours for controllers that are not part of the SNN development process. The BNS approach is thus more suited for developing behaviours for a single task.

The BNS approach did exhibit issues related to task scalability and consistency in the number of real-world evaluations required to develop effective solutions. The number of real-world evaluations required for the BNS approach to develop effective controllers for the same task varied significantly. The scalability of the BNS approach was observed to be an issue where a large number of real-world evaluations was required for complex tasks compared to simpler ones which indicated that the BNS approach could be researched further in order to improve scalability.

Little research has been conducted on furthering knowledge related to SNNs and no known research beyond this study has investigated bi-directional SNN approaches. It would be beneficial to study variations in the BNS approach for various tasks and robot morphologies not covered in this research. A more detailed discussion on the limitations and future potential of the BNS approach is given in Sections 6.4 and 6.5, respectively.

6.3 Contributions

Physics-based simulators require a significant amount of knowledge of the physics involved for the development of a simulator (Section 2.4). SNNs require little knowledge of the physics governing the robotic system and serve as an alternative to physics-based approaches to simulation. SNNs need to be pre-computed which is time-consuming and requires a large number of controller evaluations. The BNS approach developed in this research allowed for the concurrent development of controllers and SNNs during the ER process and thus eliminated the need for pre-computing SNNs. This research did not attempt to directly compare the pre-computed and BNS approach to SNN development, however, the results observed in this work indicated that in certain situations, the BNS approach could potentially speed up the SNN development process and reduce the number of real-world controller evaluations. The developed simulators are also more targeted towards the required behaviours.

Real-world robot prototypes of the BNS approach were developed and used to demonstrate the effectiveness of the proposed approach for two significantly different robot morphologies. The viability of using SNNs to simulate the behaviours of a snake-like robot was initially unknown. Pre-computed SNNs were developed in this research in order to demonstrate the viability of using SNNs to simulate snake-like robot behaviours (Sections 5.4 and 5.5). Using SNNs for a snake-like robot is significant as the approach requires little specialised knowledge of the dynamics of the complex robotic system.

The parameter comparison experiments identified the most influential parameters in the success of the approach and were able to indicate which parameter values increase the likelihood of success. It was demonstrated that there needs to be a balance between the exploration of the controller search space in order to find new solutions and the exploitation of existing solutions. The BNS approach on the given problems performed optimally with a low controller mutation rate and a high

controller population size.

A low mutation rate allows for a high level of exploitation of existing solutions while a high population size results in a greater exploration of the solution search space. Lower controller mutation rates tended to perform better, because higher values likely resulted in the controller evolution process performing a more exploratory search of the solution space and/or cause known good solutions to be distorted. A higher population size increases the degree of exploration performed during the controller evolution process and reduces the likelihood that known good solutions are lost.

The controller mutation rate was found to be the most influential parameter for the Khepera robot and was the second most influential parameter for the snake robot. The population size was the most influential parameter for the success of the snake robot. A low controller population size greatly reduced the likelihood of finding a successful solution for the snake robot, whereas this parameter was not as critical to the success of the Khepera robot. This could be due to the controller solution space of the snake robot being much more complex than that of the Khepera robot. This indicates that a more complex solution space may increase the importance of the controller population size in the success of the approach. A low mutation rate helps controllers to exploit known good solutions which in turn speeds up the simulator development process because of the targeted training data generated by these controllers. A large population size helps the controller evolution process to explore a more diverse set of solutions at a time.

6.4 Limitations

Developing SNNs may require the evaluation of many controllers on a real-world robot. Exploring the entire simulator search space may not be feasible for complex robots and there is no guarantee of finding successful solutions. During the controller evolution process, good solutions may be lost due to a changing simulator. The BNS

approach may result in a continual exploration of the behavioural search space and may be unable to focus on a particular strategy or the BNS approach may converge on a sub-optimal solution for the controller.

The BNS approach develops a specialised simulator that is targeted towards simulating certain task behaviours and is less able to simulate the behaviours of an unknown task. A concurrently developed simulator is therefore less able to simulate random behaviours than a pre-computed simulator. For multiple tasks, the BNS approach needs to be repeated for each task. SNNs are trained to simulate the behaviours of a particular robot in a certain environment and may not adapt easily to changes during the BNS approach.

6.5 Recommendations for Future Investigation

The BNS approach developed and studied in this research was kept simple in order to serve as a basis for future variations and improvements in the approach. Many other aspects of the approach, such as simulator training, controller evolution or real-world evaluations could be investigated further and methods varied in order to identify possible improvements. The current approach attempts to evolve controllers for an entire task at once but an alternative approach could evolve controllers to solve a portion of the task and build up the controller to solve the complete task.

It is unknown how well the BNS approach adapts to changes to the robot or environment during the ER process. The approach could be adapted to detect and account for any changes to the robot or environment during the ER process.

This study was limited to robots without sensors. Controllers and robots making use of sensors would be of particular interest in future research. The viability of the approach developed in this research was investigated on two different robot morphologies for specific tasks and controllers. Future work could establish the viability of the approach for various other morphologies, tasks or controllers. The ability of SNNs, and in particular the BNS approach, to scale up to complex robots

such as bipedal or quadruped robots is unknown.

Hybrid simulation approaches could be constructed where certain parts of the robotic system are simulated by using a physics-based approach and other parts are predicted using SNNs. Evaluating controllers on multiple types of simulators during the ER process could be investigated. During the early stages of the ER process, controllers could be evaluated on a physics-based simulator and behavioural data, collected during real-world evaluations, could simultaneously be used to develop SNNs that eventually replace or complement the physics-based simulator.

6.6 Summary

An important goal in ER that was explored in this research is the automatic, real-time creation of controllers and simulators with minimal human intervention or specialised knowledge. Few ER researchers have proposed effective solutions to this goal, because most simulation approaches used in ER require a great deal of specialised and manual intervention. Traditional SNNs have been shown to provide a viable alternative to traditional simulators for reducing the need for specialised knowledge or human interventions, and bi-directional approaches to controller and simulator creation have shown much promise in automating the development of physics-based simulators.

The prototypes developed in this research demonstrated that the concurrent development of controllers and SNNs during the ER process is indeed viable. Controllers were successfully developed for two very different robot morphologies. The Khepera prototype served as an initial proof-of-concept and the snake robot was used investigate the scalability of the new approach. This research demonstrated that pre-computed SNNs can be used to simulate snake-like locomotion and later also demonstrated that controllers and SNNs could be developed concurrently for the snake robot during the ER process. Intuitively it is difficult to develop a physics model for the snake robot that could adequately simulate snake locomotion well

enough to develop viable controllers. Developing a simulator without the need for any specialised knowledge or lengthy simulator development process was therefore a significant contribution.

The BNS approach was shown to successfully scale well from a simple differentially-steered robot to a complex snake-like robot with many degrees of freedom. These are encouraging results, because they indicate that SNNs and in particular, the BNS approach could scale well for more complex robots. The extensive parameter comparison study conducted as part of this research was able to identify that the controller mutation rate and population size were the most influential parameters tested. The BNS approach favoured a low mutation rate for the exploitation of known good solutions which helped the simulator focus on learning only those required behaviours, while a high population size increased the diversity of solutions during any given generation which helped to alleviate any issue related to a changing simulator.

Bibliography

P. Abbeel, M. Quigley, and A. Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine Learning*, pages 1–8. ACM, 2006.

D. Beasley, R. Martin, and D. Bull. An overview of Genetic Algorithms: Part 1. Fundamentals. *University Computing*, 15(2):58–69, 1993.

J. Bongard. Evolutionary robotics. *Communications of the ACM*, 56(8):74–83, 2013.

J. Bongard and H. Lipson. Automated damage diagnosis and recovery for remote robotics. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3545–3550. IEEE, 2004.

J Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *Evolutionary Computation, IEEE Transactions on*, 9(4):361–384, 2005.

J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, December 2006a.

J. Bongard, V. Zykov, and H. Lipson. Automated synthesis of body schema using multiple sensor modalities. In *Proceedings of the International Conference on the Simulation and Synthesis of Living Systems (ALIFEX)*, 2006b.

R. Brooks. Artificial life in real robots. *Artificial Intelligence*, 48:3–10, 1992.

S. Carpin, T. Stoyanov, Y. Nevatia, M. Lewis, and J. Wang. Quantitative assessments of usarsim accuracy. In *In Proceedings of Performance Metrics for Intelligent Systems (PerMIS) Workshop*, 2006.

A. Cully, J. Clune, D. Tarapore, and J. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

R. De Nardi. *Automatic Design of Controllers for Miniature Vehicles through Automatic Modelling*. PhD thesis, University of Essex, 2010.

R. De Nardi and O. Holland. Coevolutionary modelling of a miniature rotorcraft. In *10th International Conference on Intelligent Autonomous Systems (IAS10)*, pages 364–373, 2008.

K. Dowling. *Limbless locomotion: Learning to crawl with a snake robot*. PhD thesis, NASA, 1996.

G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan. Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51. IEEE, 2011.

A. Engelbrecht. *Computational intelligence: An introduction*. John Wiley & Sons, 2007.

D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. *From animals to animats*, pages 421–430, 1994.

D. Floreano, P. Husbands, and S. Nolfi. Evolutionary robotics. *Springer handbook of robotics*, pages 1423–1451, 2008.

J. Grefenstette and C. Ramsey. An approach to anytime learning. In *Machine Learning: Proceedings of the Ninth International Conference*, pages 189–195, 2014.

Heaton Research. Official Encog Site. `http://www.heatonresearch.com/encog`, 2014. Accessed: December 2014.

D. Hu, J. Nirody, T. Scott, and M. Shelley. The mechanics of slithering locomotion. *Proceedings of the National Academy of Sciences*, 106(25):10081–10085, 2009.

N. Jakobi. *Minimal simulations for Evolutionary Robotics*. PhD thesis, University of Sussex, 1998.

N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in Evolutionary Robotics. In *Advances in artificial life*, pages 704–720. Springer, 1995.

K-Team. Khepera III. `http://www.k-team.com/mobile-robotics-products/khepera-iii`, 2014. Accessed: November 2014.

A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji. Automatic locomotion design and experiments for a modular robotic system. *Mechatronics, IEEE/ASME Transactions on*, 10(3):314–325, 2005.

S. Kamio and H. Iba. Evolutionary construction of a simulator for real robots. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 2202–2209. IEEE, 2004.

D. Keymeulen, M. Iwata, Y. Kuniyoshi, and T. Higuchi. Online evolution for a self-adapting robotic navigation system using evolvable hardware. *Artificial Life*, 4(4):359–393, 1998.

S. Koos, J. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *Evolutionary Computation, IEEE Transactions on*, 17(1):122–145, 2013.

J. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

T. Laue, K. Spiess, and T. Röfer. SimRobota general physical robot simulator and its application in robocup. In *RoboCup 2005: Robot Soccer World Cup IX*, pages 173–183. Springer, 2006.

J. Lee. Hacking the Nintendo Wii remote. *Pervasive Computing, IEEE*, 7(3):39–45, 2008.

T. Lee, U. Nehmzow, and R. Hubbold. Mobile Robot Simulation by Means of Acquired Neural Network Models. In *ESM*, pages 465–469, 1998.

T. Lee, U. Nehmzow, and R. Hubbold. Computer simulation of learning experiments with autonomous mobile robots. *Proceedings of TIMR*, 99, 1999.

D. Lizotte, T. Wang, M. Bowling, and D. Schuurmans. Automatic Gait Optimization with Gaussian Process Regression. In *IJCAI*, volume 7, pages 944–949, 2007.

H. Lund. Co-evolving control and morphology with LEGO Robots. In *Morpho-functional Machines: The New Species*, pages 59–79. Springer, 2003.

H. Lund and O. Miglino. From simulated to real robots. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 362–365. IEEE, 1996.

A. Maren, C. Harston, and R. Pap. *Handbook of neural computing applications*. Academic Press, 2014.

M. Matarić and D. Cliff. Challenges in evolving controllers for physical robots. *Robotics and autonomous systems*, 19(1):67–83, 1996.

K. Melo, M. Hernandez, and D. Gonzalez. Parameterized space conditions for the definition of locomotion modes in modular snake robots. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 2032–2038. IEEE, 2012a.

K. Melo, L. Paez, and C. Parra. Indoor and outdoor parametrized gait execution with modular snake robots. In *2012 IEEE International Conference on Robotics and Automation*, 2012b.

O. Miglino, K. Nafasi, and C. Taylor. Selection for wandering behavior in a small robot. *Artificial Life*, 2(1):101–116, 1994.

O. Miglino, H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial life*, 2(4):417–434, 1995.

R. Moeckel, Y. Perov, A. Nguyen, M. Vespignani, S. Bonardi, S. Pouya, A. Sproewitz, J. van den Kieboom, F. Wilhelm, and A. Ijspeert. Gait optimization for roombots modular robots - matching simulation and reality. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3265–3272. Ieee, 2013.

J. Mouret, S. Koos, and S. Doncieux. Crossing the reality gap: a short introduction to the transferability approach. In *In Proceedings of the ALIFE workshop "evolution in physical systems"*, 2012.

S. Murata and H. Kurokawa. Self-reconfigurable robots. *Robotics & Automation Magazine, IEEE*, 14(1):71–78, 2007.

S. Nakamura and S. Hashimoto. Hybrid learning strategy to solve pendulum swing-up problem for real hardware. In *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*, pages 1972–1977. IEEE, 2007.

S. Nakamura, R. Saegusa, and S. Hashimoto. A hybrid learning strategy for real hardware of swing-up pendulum. *Journal of Advanced Computational Intelligence & Intelligent Informatics (JACIII)*, 11(8), 2007.

U. Nehmzow, D. Kerr, and S. Billings. Accurate robot simulation. Technical Report ACSE Research Report no. 992, University of Sheffield, 2009.

Nintendo. Wii Official Site at Nintendo. `http://wii.com`, 2014. Accessed: November 2014.

S. Nolfi and D. Parisi. Evolving non-trivial behaviors on real robots: an autonomous robot that picks up objects. In *Topics in Artificial Intelligence*, pages 243–254. Springer, 1995.

OpenCV. Official OpenCV Site. `http://opencv.org`, 2015. Accessed: July 2015.

J. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. Watson. Evolutionary techniques in physical robotics. In *Evolvable Systems: from biology to hardware*, pages 175–186. Springer, 2000.

D. Pratihar. Evolutionary robotics - A review. *Sadhana*, 28(6):999–1009, 2003.

C. Pretorius. *Artificial Neural Networks as simulators for behavioural evolution in Evolutionary Robotics*. Masters thesis, Nelson Mandela Metropolitan University, 2010.

C. Pretorius, M. du Plessis, and C. Cilliers. Towards an Artificial Neural Network-based simulator for behavioural evolution in Evolutionary Robotics. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 170–178. ACM, 2009.

C. Pretorius, M. du Plessis, and C. Cilliers. Simulating robots without conventional physics: A neural network approach. *Journal of Intelligent & Robotic Systems*, 71(3-4):319–348, 2013.

C. Pretorius, M. du Plessis, and J. Gonsalves. A comparison of Neural Networks and physics models as motion simulators for simple robotic evolution. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 2793–2800. IEEE, 2014.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation

learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

H. Schwefel. *Evolution and optimum seeking: the sixth generation*. John Wiley & Sons, Inc., 1993.

O. Shmakov. Snakelike robots locomotions control. *Mechatronics–Foundations and Applications*, 2006.

R. Smith. Open Dynamics Engine, 2007. URL `http://www.ode.org`. Accessed: November 2015.

D. Sofge, M. Potter, M. Bugajska, and A. Schultz. Challenges and opportunities of evolutionary robotics. In *Proceedings of the Second International Conference on Computational Intelligence*. Robotics and Autonomous Systems, 2003.

J. Togelius, R. De Nardi, H. Marques, R. Newcombe, S. Lucas, and O. Holland. Non-linear dynamics modelling for controller evolution. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 324–333. ACM, 2007.

S. Wittmeier, M. Jäntsch, K. Dalamagkidis, and A. Knoll. Physics-based modeling of an anthropomimetic robot. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4148–4153. IEEE, 2011.

J. Zagal and J. Ruiz-del Solar. Combining simulation and reality in Evolutionary Robotics. *Journal of Intelligent and Robotic Systems*, 50(1):19–39, March 2007. ISSN 0921-0296.

Appendix A

IEEE (SSCI) 2015 Paper

The pilot study for investigating the viability of using SNNs for a complex snake-like robot was conducted and presented. An initial snake robot prototype was developed and behavioural data was collected. This behavioural data was subsequently used to train a set of SNNs that were used to evaluate controller fitnesses during an ER process. Accuracy of the SNNs was assessed and the transferability of controllers validated on a real-world robot.

This study was presented in a conference paper that was accepted for presentation at the 2015 IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2015). The paper follows.

Evolving Snake Robot Controllers using Artificial Neural Networks as an Alternative to a Physics-Based Simulator

Grant W. Woodford
Department of Computing Sciences
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Email: grant.woodford@nmmu.ac.za

Mathys C. du Plessis
Department of Computing Sciences
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Email: mc.duplessis@nmmu.ac.za

Christiaan J. Pretorius
Department of Mathematics and Applied Mathematics
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Email: cpretorius@nmmu.ac.za

***Abstract*—Traditional simulators can be complex, time-consuming and require specialized knowledge to develop while still being unable to adequately model reality. Artificial Neural Networks (ANNs) can be trained to simulate real-world robots and therefore serve as an alternative to traditional approaches of robot simulation during the Evolutionary Robotics (ER) process. ANN-based simulators require little specialized knowledge and can automatically incorporate many real-world peculiarities. This paper reports a simulator that consisted of ANNs which were trained to predict changes in the position of a real-world snake-like robot. Navigational behaviours were evolved in simulation and subsequently verified on the real-world robot. This paper demonstrated that ANNs are a viable alternative to traditional simulators for evolving controllers for snake-like robots.**

I. Introduction

The field of Evolutionary Robotics (ER) seeks to automate the development of intelligent control structures for robotic systems using Evolutionary Computing approaches [1]. The manual development of robot controllers becomes infeasible as robots, environments and tasks increase in complexity [2]. ER has been shown to automatically evolve many robot behaviours, such as path following, inverted pendulum stabilization, light following and obstacle avoidance [3], [4].

In ER, many controllers are evaluated and their relative performances quantified in order to evolve better controllers. The evaluation of many controllers on a real-world robot is time-consuming and can damage hardware [5]. These issues can be overcome through the use of simulators as an alternative to real-world evaluations [2]. Traditional simulators are physics-based or modelled on empirically collected data [3]. These simulators can be time-consuming and complicated to develop because it may require the use of complex physics-based models and/or the gathering of large amounts of experimental data.

It has been shown that alternatively, simulators can be constructed using ANNs that are trained to predict robot behaviours using experimentally collected data [3], [4], [6], [7]. In this paper, ANN-based simulators will simply be referred to as Simulator Neural Networks (SNNs). The use of SNNs have shown much promise as an alternative to traditional approaches to simulation during the ER process [4]. SNNs have been shown to be computationally efficient, require little specialized knowledge, possess good prediction accuracy, noise-tolerance and generalization abilities in modelling of certain robot phenomena [7]. Previous work has mainly focused on simple robots, therefore this paper aims to investigate the use of SNNs during the ER process on a complex robot.

This paper is structured as follows: Section II-A describes the ER process while Section II-B addresses the use of simulators during the ER process. Related work on snake-like robots is discussed (Section II-C) while the experimental robot used in this paper is considered (Section III). The simulator developed in this paper is proposed in Section III-A1. The robotic controller and method of behavioural tracking used in this paper are discussed in Sections III-B and III-C respectively. The experimental procedure followed for simulator training and the subsequent validation of the proposed simulator are addressed in Sections IV-A and IV-B. The results determined that the proposed simulator is indeed viable, with the simulator's accuracy and real-world validation experiments presented in Sections V-A and V-B respectively. These results are discussed in Section VI and finally conclusions are drawn and possible future work is discussed (Section VII).

II. Background

A. Evolutionary Robotics

In ER, robotic controllers are evolved to develop behaviours using Genetic Algorithms (GA) [1]. To develop the appropriate behaviours, a population of encoded candidate controllers is created. Each controller's fitness relative to each other is determined based on how well the controller exhibits the target behaviour on the experimental robot or in simulation, after which a new generation of controllers is created to replace the previous generation. The new generation is generated using reproduction operators (crossover and mutation) between the controllers of the previous generation.

Controllers with a higher fitness have a greater probability of being chosen to produce offspring for the new generation.

978-1-4799-7560-0/15/$31

During the crossover process, genetic material between parent controllers are combined and passed onto the new generation. The mutation operator can then be applied, causing small random perturbations in controllers which allows for a more diverse controller population. This process is repeated for a large number of generations and controllers ideally converge towards the target behaviour. The end result of the ER process is an optimized controller that is validated on a real-world robot.

A major issue during the ER process is the evaluation of a large number of controllers for fitness determinations (Section I). Determining the fitnesses of many controllers on a real-world robot would be time-consuming and financially costly. Controller fitness evaluations can therefore be done in simulation to speed up the process.

B. Evolution in Simulation

Simulators are able to overcome issues inherent in real-world fitness evaluations. Controller evolution can explore the search space more rapidly in simulation than it would be possible in reality [15]. However, as previously mentioned, the design and construction of traditional physics-based simulators can become time-consuming and complicated [7]. Much research in ER is concerned with overcoming the difficulties inherent in utilizing simulators effectively [16]. Challenges in simulator design can be inaccuracies and/or over-simplification in the modelling of reality [7].

Over-simplification or inaccuracies in simulations may result in controllers relying on peculiarities that exist only in simulation but are non-existent in reality which results in behaviours evolved in simulation not transferring well to reality, commonly referred to as the *reality gap* problem [17]. Over-simplification can be avoided through the use of highly accurate simulators. However, even highly accurate simulators cannot perfectly model reality and will inevitably contain inaccuracies [18]. Additionally, highly accurate simulators are often computationally expensive [19]. Simulators ideally need to provide highly accurate representations of reality whilst not being too computationally expensive to use.

There are currently few alternatives to physics-based simulators in use by ER researchers. The notion of using SNNs as an alternative to traditional simulators has been investigated by few researchers [6], [20]. As previously mentioned (Section I), SNNs are computationally efficient, accurate, relatively simple to construct and potentially provide an effective alternative to physics-based approaches. The training of SNNs requires the evaluation of many randomized behaviours on a real-world robot and the collection of this behavioural data. This behavioural data is then used to train SNNs to predict robot behaviours.

SNNs have been effectively used during the ER process for tasks such as path following, obstacle avoidance, light approaching behaviour and inverted pendulum stabilization [3], [4], [6], [7]. Researchers have also shown that SNNs can simulate the dynamics of the pendulum swing-up problem [20].

C. Snake Robots

The ER process for snake-like robots is mostly carried out in simulation using physics-based approaches to estimate controller fitness [8]–[11]. A physics-based simulator and GA approach has been used for shape transformation planning to achieve stable, smooth transitions between snake locomotion modes [12]. Additionally, modular robotics research has been conducted using snake-like configurations and a GA to evolve controllers in a physics-based simulator [13], [14].

Snake-like robots are constructed by chaining together several independent actuators, each actuator commonly having one degree of freedom [13], [21], [22]. These snake robots are capable of both biological and non-biologically inspired locomotion modes.

Biologically inspired snake locomotion modes can be broadly classified into lateral undulation, slide-pushing, rectilinear motion, concertina, side-winding and various other forms [23]. Lateral undulation is the most common form where all parts of the body move in a wave-like pattern [23]. However, lateral undulation is not suited to smooth, low friction surfaces [24]. Side-winding locomotion has a sine-like wave while maintaining only two static points of contact with the ground at any time. Side-winding is more suited for low friction surfaces [24]. There are also non-biologically inspired locomotion modes, namely rolling where the snake rolls side over side and flapping motions where the robot flaps both of its ends across the ground [23].

One method of generating the above mentioned locomotion modes is the use of parametrized equations which generate the appropriate joint angles on the robot. Melo *et al.* [21] have made use of parametrized Equations (1) and (2) which are based on sinusoidal motion on two axes. These equations define $\phi(n,t)$ as the angle of the n^{th} joint at discrete time steps t. Snake joints were numbered, starting at zero from front to back, with evenly numbered joints moving the snake laterally and odd ones moving the snake vertically. If the snake rolled on its side, then the evenly numbered joints moved vertically and oddly numbered joints moved laterally which similarly switched the angles used by Equation (1). The terms $O_{lateral}$ and $O_{vertical}$ are the offsets of the lateral and vertical joints respectively. The offset terms define the central angle value of their respective waves. Terms $A_{lateral}$ and $A_{vertical}$ refer to the wave amplitudes of the lateral and vertical joint angles respectively. The parameters ω and γ are respectively the spatial and temporal components of the sine wave moving through the robot. The α term is the phase offset between the lateral and vertical sine waves.

$$\phi(n,t) = \begin{cases} O_{lateral} + A_{lateral} \cdot sin(\theta + \alpha), n = even \\ O_{vertical} + A_{vertical} \cdot sin(\theta), n = otherwise \end{cases} \quad (1)$$

$$\theta = \omega n + \gamma t \quad (2)$$

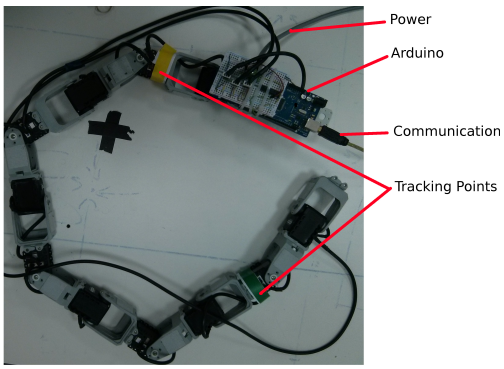


Fig. 1. Robot Morphology

III. EXPERIMENTAL ROBOT

A custom designed snake-like robot was developed for the experimental work. Similarly constructed snake-like robots have also been used by other researchers [14], [22].

This robot was chosen due to its simple construction and relatively low cost. The morphology of the robot used for experimental work is shown in Figure 1. An array of twelve Dynamixel AX-12 servo motors joined together with links formed the robot's body. Evenly numbered joints moved the robot laterally while the odd joints moved the robot vertically. The length of the robot was 114cm and its width and height were 5cm. The servos were controlled by an Arduino Mega micro-controller which received joint angles from a computer using serial communication.

The servos were powered using a tethered connection to the robot. A tethered approach was chosen due to the extra weight batteries would add and to eliminate the need for monitoring battery power levels. Near each end of the robot was a yellow or green tracking marker which was used for camera based tracking.

No snake-like skins or mechanisms such as wheels were used to allow for the directional friction required for forward locomotion seen in many biological snakes. This resulted in the robot having difficulty with forward locomotion and having to either side-wind, strafe laterally, perform helix-like rolling or flapping motions. The robot was not fitted with any sensors.

A. IMPLEMENTATION DETAILS OF SIMULATOR AND EXPERIMENTAL SETUP

The design and construction of the developed simulator is addressed in Section III-A1. Details of the controller used and method of behavioural tracking are discussed in Sections III-B and III-C respectively.

1) Proposed Approach for using Simulator Neural Networks: A set of twelve joint angles represented a single command. When a command was sent to the robot, all of the robot's joints simultaneously positioned to the assigned angles. A cycle consisted of thirteen sequential commands which when evaluated, started and ended on the same joint angle set. During the evaluation of a cycle, commands were sent sequentially to the robot and upon reaching all the assigned joint angles of the given command, the robot moved onto the next command.

Equation (1) is used to generate a sequence of commands that perform cyclical behaviours when evaluated on a robot. The robot's change in position after the evaluation of a given cycle depends on the parameter settings used by Equations (1) and (2). SNNs take as input those parameter settings and subsequently predict the change in the robot's position on the planar operating surface at the end of the given cycle. The simulator also predicts whether cycles will fail and cause the robot to tip over, roll, reach torque limits or collide with itself.

The creation of SNNs and controllers using the ER process is achieved as follows:

1) Parameter settings for Equations (1) and (2) are randomly generated using a uniform distribution and used to generate cycles that are performed on the real-world robot. As a result of these commands, the robot moves around the environment and data is collected using motion tracking techniques.
2) When sufficient data has been collected, it is used to train SNNs to predict the real-world robot's behaviours. Data collection is concluded when the trained SNNs are able to evolve adequately transferable controllers that complete the goal task.
3) The trained SNNs are used to determine the fitness of candidate controllers during controller evolution.
4) At the end of the ER process, the fittest controller in simulation is validated using the real-world robot.

SNNs were trained to predict the change in the position of the tracked markers on the robot for a given cycle. The robot had two local coordinate systems with the origins located at the centre of each tracked marker and y directions taken from the yellow to green tracked markers. The change in the x and y-coordinates for the yellow tracked marker for a given cycle was represented by Δx_1 and Δy_1 respectively. Similarly, the changes in the x and y-coordinates for the green marker were represented by Δx_2 and Δy_2 respectively. The simulator also predicted whether or not the robot remained upright and did not collide with itself or reach any torque limits during a given cycle.

The simulator developed in this study consisted of five separate Feed Forward Neural Networks (FFNNs) (Figure 2), one for each of the x and y directions for each tracked marker and another to predict failed cycles. A previous study determined that separate FFNNs could produce more accurate results than a single FFNN [6]. Each FFNN took as input the values of $A_{lateral}$, $A_{vertical}$, α and ω that were used by Equation 1 to generate the cycle. The offset and γ terms were kept constant and not used as inputs. Sigmoid activation functions were used for all FFNN neurons and each FFNN had a single hidden layer of 100 neurons.

In order to evolve robust controllers that were able to cross the *reality gap* (Section II-B), noise was injected into the ER process during controller evaluations. Controllers were

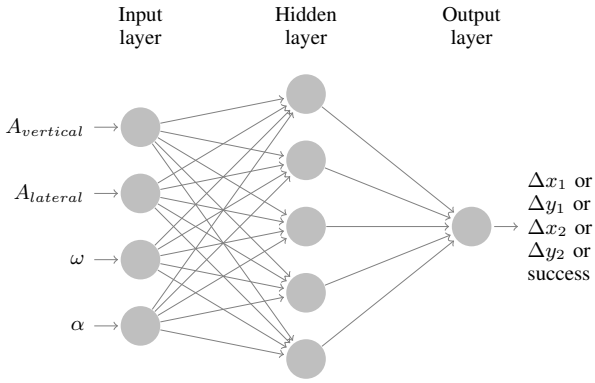


Fig. 2. Simulator Neural Networks

evaluated ten times in simulation and the average fitness was used. The distribution used for noise injection was Gaussian with a mean of zero and standard deviations of 10cm and 5cm for the $\Delta x_{1,2}$ and $\Delta y_{1,2}$ displacements respectively. These standard deviations were based on the observed training data errors.

B. Robot Controller

A controller consisted of a sequential list of four different parameters settings that were used to generate cycles using Equations (1) and (2). Each cycle in the list was repeated up to four iterations when evaluated. It was observed through manual experimentation that the chosen controller morphology is more than sufficient to accomplish the required behaviours. A single controller could consist of between four to sixteen cycles.

The γ term determined the number of commands contained in a given cycle. The set of joint angles for all cycles had to begin and end on the same set of angles which is exactly one period of the sine wave. The time-steps of t for every cycle went from zero to the number of commands per cycle plus one. The commands per cycle were fixed at thirteen which required that the γ term remained constant at $2\pi/12$. The offsets $O_{lateral}$ and $O_{vertical}$ determined the central angle value for the wave which was assumed to remain constant at zero. Non-zero offsets typically help steer the robot in a particular direction. Amplitudes $A_{lateral}$ and $A_{vertical}$ had a range of between 0 and $\pi/2$. The α and ω parameters were within the sine function, therefore their values ranged from 0 to 2π. The reason the number of commands per cycle, the offsets and γ terms were chosen to remain constant was to reduce the controller search space which in turn reduced the amount of training data the simulator required to perform adequately.

The vertical amplitude $A_{vertical}$ was chosen so that it was less than the lateral amplitude $A_{lateral}$ in order to reduce the number of cycles that failed to remain upright. Even with this restriction in place, many parameter sets could result in the robot tipping over or rolling. Ensuring that the robot always remained upright and stable during evaluations was important for the simulator predictions. The simulator was trained to only predict behaviours of upright cycles.

The angular velocity of the robot's joint movements was kept constant. The angular velocity was experimentally chosen to reduce slippage on the smooth operating surface. Depending on the locomotion mode, the degree of slippage varied greatly. Due to the flexibility of Equation 1, many different locomotion modes were possible, even modes that resulted in collisions with the robot itself or caused joint torque limits to be reached.

C. Motion Tracking

A roof-mounted camera was used to track robot behaviours, namely the change in position of the tracked markers on the robot for a given cycle. A camera based tracking approach was chosen over manual data acquisition methods in order to speed up the process and eliminate human error. An open source computer vision library called OpenCV [25] was used for the camera based tracking software.

Images from the tracking camera were used to locate the pixel coordinates of the tracked markers. The centre of each marker was determined and converted into real-world coordinates. Calibration techniques were employed and distortions were removed from the captured images before use. The automated tracking process was not used to identify if the robot tipped over, rolled, collided with itself or if the torque limits were reached, therefore these behaviours were manually noted during data acquisition.

IV. EXPERIMENTAL PROCEDURE FOR VALIDATING SIMULATOR

Experimental data from the real-world robot was acquired and used to train the simulator (Section IV-A). Once trained, the simulator was used to evolve robotic controllers to perform a navigational task on the real-world robot (Section IV-B).

A. Simulator Training

To obtain sufficient data for SNN training, randomly generated cycles were evaluated on the real-world robot and behavioural data was collected. The change in position for each cycle was recorded by the roof-mounted camera. The robot operated on a working surface of dimensions 2.7m by 1.85m.

Training data was experimentally acquired from 400 randomly generated cycles. This training data was used to train an initial simulator and the worst twenty training data cycles that had the largest difference between their expected versus simulated displacements were subsequently selected for each of the SNNs. These poor performers were re-evaluated and each one used to generate four additional cycles with noise, generating 400 additional training data cycles. The SNNs were then re-trained utilizing all training data. A bidirectional approach was taken to assist the simulator predictions to become more accurate and stable. These SNNs were used to evolve controllers during the ER process and twenty of the worst performing cycles from these controllers were taken and used to generate a further 100 training data patterns. The

final SNNs were then trained using all available training data generated.

The SNNs were implemented using an open source machine learning library called Encog [26]. SNNs were trained using Resilient Backpropagation for 12000 iterations or just before over-fitting occurred. The sizes of the training data set and verification data of the stable cycles were 720 and 76 respectively. The sizes of the training data set and verification data set of both the stable and unstable cycles were 820 and 80 respectively. The data for the stable cycles were used to train the SNNs to predict the changes in the x and y direction of both tracked markers for a given cycle. The data for the stable and unstable cycles were used to train another SNN to predict whether a given cycle would be stable or unstable.

B. Validation Experiment

Due to this investigation being an initial proof-of-concept prototype, the navigational task was deliberately chosen to be simple in order to investigate the viability of the approach. The required navigational behaviour is shown in Figures 7 to 9. Three squares, each 40cm wide were placed in three quadrants of the operating surface and placed 20cm apart from each other. The robot was placed with the yellow marker on the origin and facing eastwards. The task was considered completed when the robot traversed all the blocks in a specific order (top right, bottom left, top left).

During the ER process, controllers were evaluated in simulation, generating a list of points which represented the simulated path. The fitness values assigned to each controller were determined using Algorithm 1. This algorithm took as input, the simulated path and the list of goal regions. The fitness was calculated based on the number of the goal regions reached in their specified order. A controller's fitness was penalized for containing positions outside the bounds of the working surface. If a position was reached from an unstable cycle, the fitness was also penalized. The fitness was further penalized for each goal region not reached.

Controllers consisted of encoded parameters for the variable terms in Equations (1) and (2). The controller evolution settings are given in Table I. An initial population of 400 controllers was generated randomly from a uniform distribution. During uniform cross-over operations, two parents were selected using tournament selection (using a tournament size of 20 parents). Child controllers were comprised of approximately 20% genetic material from one parent and 80% from the other. The probability that controller parameter values were mutated by a random amount was 10%. The ER process proceeded until the fittest controller was sufficiently able to complete the task in simulation. Upon completion, the final controller was evaluated on the real-world robot.

Algorithm 1: Controller fitness evaluation

Data:
$positions \leftarrow$ Simulated path list
$goalpoints \leftarrow$ List of goal regions
Result: Fitness of controller
$sum \leftarrow 0$
$curGP \leftarrow 0$ ▷ Current goal point
$penalty \leftarrow$ a large constant
for *each position in positions* **do**
  if *position out of bounds* **then**
    $sum \leftarrow sum + penalty$
  end
  if *position reached* $goalpoints[curGP]$ **then**
    $curGP \leftarrow curGP + 1$
  end
  if *position reached from failed cycle* **then**
    $sum \leftarrow sum + penalty$
  end
end
$missedGoalpoints \leftarrow length(goalpoints) - curGP - 1$
$sum \leftarrow sum + penalty \times missedGoalpoints$
$fitness \leftarrow 1/sum$
return *fitness*

TABLE I
PARAMETERS FOR CONTROLLER EVOLUTION

Population Size	400
Initialization	Random from a uniform distribution
Selection	Tournament (size 20)
Crossover Method	Uniform
Mutation Probability	10%
Mutation Method	Random Component Perturbation

V. RESULTS

Section V-A discusses the accuracy of the simulator developed in Section IV by analysing the predicted versus expected change in the robot's position using the verification data set. Section V-B presents the results of the optimized controllers run on a real-world robot and compares the behaviours to simulation.

A. Simulator Accuracy

Figures 3 to 6 demonstrate the predicted versus expected change in position of the tracked markers for the verification data set. Every dot represents the simulated versus real-world change in position of the robot for the specific coordinate axis and for the given verification cycle. The verification data set was not presented to the simulator during training and was therefore used to determine the accuracy and generalization abilities of the trained simulator.

For Figures 3 to 6, linear regression lines were fitted. All the y-intercepts of the regression lines were close to zero. The R-squared values for Figures 3 and 5 were 0.84 and 0.75 respectively. This indicted that the simulator could adequately model movement along the x direction of the robot. The regression line slopes were 0.85 and 0.9 for Figures 3 and 5 respectively and both y-intercepts close to zero, which indicated that the simulator modelled the real-world fairly well. The R-squared values, 0.17 and 0.05 for Figures 4 and 6 respectively indicted that the simulator could not adequately model movement along the y direction of the robot. The slopes for Figures 4 and 6 were 0.28 and 0.25 respectively, which indicated that the simulator and real-world did not closely relate for those movements.

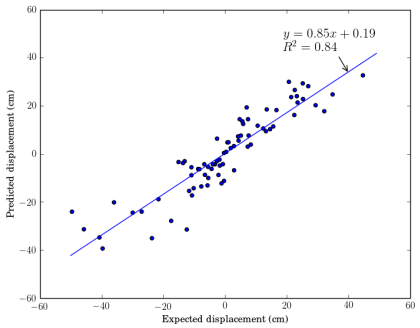


Fig. 3. Yellow tracking point's predicted versus expected displacement in the x-direction

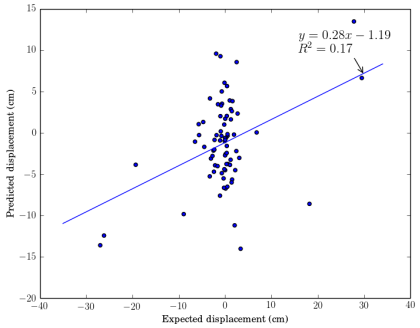


Fig. 4. Yellow tracking point's predicted versus expected displacement in the y-direction

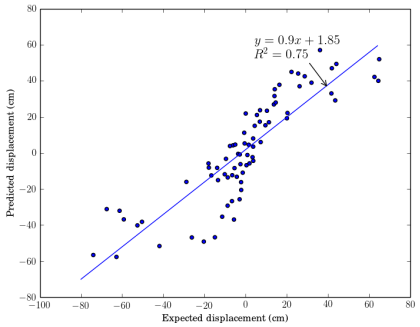


Fig. 5. Green tracking point's predicted versus expected displacement in the x-direction

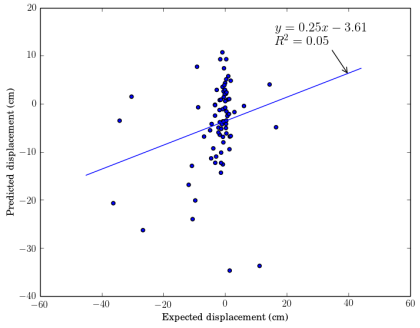


Fig. 6. Green tracking point's predicted versus expected displacement in the y-direction

The simulator was trained to predict the success or failure of any given cycle. The patterns used to verify the success or failure of a cycle contained 19 failed cycles and 63 successful cycles. The percentage of verification patterns that the simulator correctly predicted to succeed or fail was 85%. The simulator relied greatly on only predicting behaviours of successful cycles which meant that fitness estimations were inaccurate for controllers that contained any failed cycles. The percentage of verification patterns that the simulator correctly predicted to fail was 58% and the percentage of cycles that were correctly predicted to succeed was 94%.

Predictions for the robot's movement in the y direction were a great deal less accurate than that along the robot's x direction. The expected values for the robot's y directions tended to be close to zero. This was possibly due to the smooth operating surface and lack of directional friction properties seen in biological snakes or snake-like robots using wheels. These results possibly indicate that much of the movement in the y direction were close to random and that the main strategy used by the robot for locomotion was in the x direction.

The results provided a reliable indication as to the accuracy of the developed simulator and whether enough training patterns had been collected. Predicting the robot's change in positions is indeed possible which is significant considering the complexity of the robotic system.

B. Validation Results

The ideal paths obtained by validation controllers in simulation were compared to their real-world paths. These results are shown in Figures 7 to 9. The dashed lines represent the simulated path of the validation controller and the solid lines represent the real-world paths followed by the robot. The location of the robot was determined to be midway between the tracked markers. For each validation controller, three real-world evaluations were performed.

There were twelve independent trial runs, of which only the best three were presented in this paper. Three trials were excluded due to the robot moving out of bounds of the operating surface. Another three trials were excluded due to the robot rolling during transitions between cycles. One trial contained a failed cycle and two trials were left out due to poor transference to the real-world robot.

Evaluated cycles exhibited many types of locomotion modes, such as lateral strafing, side-winding, helical movements, rolling, flapping and many others. Certain behaviours, such as flapping were observed to be unable to transfer well to reality whereas other behaviours such as side-winding showed better transference to reality. Transitions between different cycles could also cause stability and accuracy problems.

The controller search space was large and often contained regions where small changes to a controller could result in large changes in behaviour. For each trial, the number of cycles used, percentage of goal regions reached, average displacement between the final positions in simulation and reality and lastly the number of controller evolution generations of the validation controllers are given in Table II. Even though

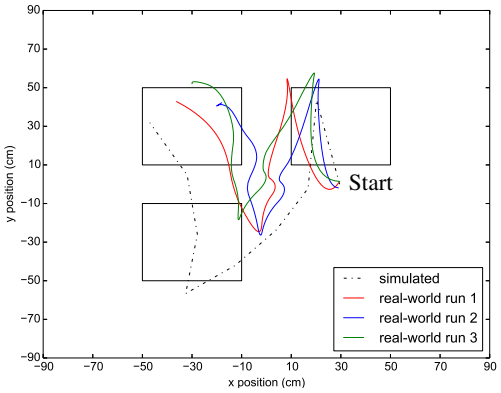


Fig. 7. Experimental run, trial 1

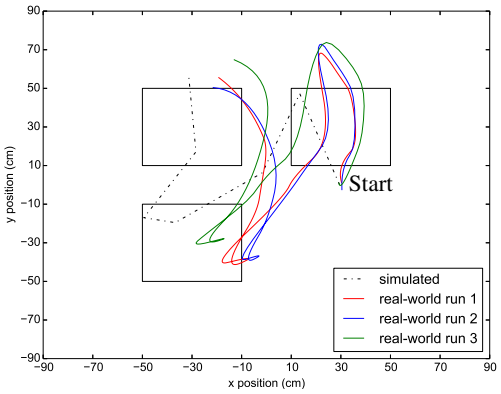


Fig. 8. Experimental run, trial 2

the navigational task was relatively simple, controllers needed many cycles to complete the task.

TABLE II
TRIAL RUN DETAILS

Trial number	Cycles	Goal regions reached	Final displacement	Controller generations
Trial 1	8	89%	21cm	200
Trial 2	6	78%	16cm	100
Trial 3	8	100%	53cm	100

The simulated and real-world controller behaviours were similar for the three trials (Figures 7 to 9). Inaccuracies between the simulator and real-world could be due to many reasons, such as path divergence over time due to an accumulation of errors. The simulator was simplified and did not predict changes in position due to transitions between cycles.

The first cycle in trial 1 (Figure 7) moved the robot into the initial goal region. The second cycle reversed the robot's direction towards origin of the working space. The slight shift eastwards around the origin was the result of the robot's shift in position due to switching to the third cycle. The third cycle was repeated three times and showed poor transference to reality which was possibly due to the lack of training data

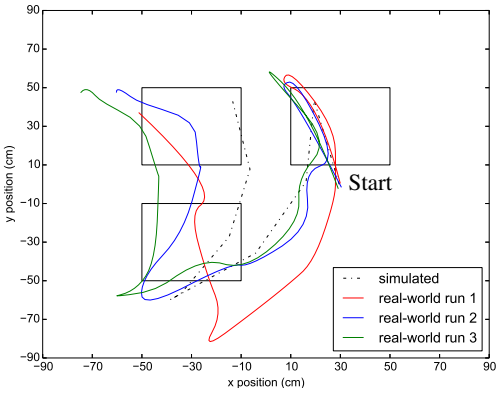


Fig. 9. Experimental run, trial 3

patterns near that region of the cycle search space. The last distinct cycle was repeated three times and brought the robot up towards the last goal region.

The first cycle in trial 2 (Figure 8) moved the robot into the initial goal region and overshot towards the right when compared to the predicted path. The next two cycles moved the robot towards and mostly into the second goal region, only one of the real-world runs missed this goal region. The fourth cycle which was supposed to move the robot westward however resulted in the real-world robot shifting position in place. The last two cycles formed a curved path towards the third goal region which closely matched the predicted behaviour.

The first cycle in trial 3 (Figure 9) moved the robot over the first goal region and overshot it slightly towards the left. The next three identical cycles brought the robot towards and just below the second goal region. The fifth cycle moved the robot slightly closer towards the second goal region and the last three cycles moved the robot over the second and third goal regions.

VI. DISCUSSION

The use of SNNs to estimate controller fitness during the ER process is indeed feasible. It was possible to use SNNs to evolve robot behaviours for a complex robot performing a simple navigational task. A direct comparison between the effectiveness of SNNs and physics-based approaches was infeasible for this limited investigation.

From the results shown in Section V-A, it was observed that changes in position of the yellow marker could be modelled more accurately than that of the green marker. This could be due to the increased friction caused by the added weight of the Arduino and tether. The increased friction could cause the robot to exhibit less slippage. An environment or robot with different frictional properties could possibly yield better results.

The developed simulator was able to predict the general behaviour of the robot. It was also significant that behaviours could be evolved for a complex robot to navigate blindly in its environment with no feedback. This was noteworthy

considering the many sources of errors. Possible sources of errors included inaccuracies in motion tracking, inconsistencies in motor function, slippage on the operating surface and errors due to the simplifying assumptions of the simulator. Poor transference could also be due to insufficient training patterns around certain portions of the cycle search space. Much work needs to be done devising better training data sampling strategies.

The complexities in the search space were significant, where small changes in cycle parameter settings could result in significant differences in behaviour. Cycles that did not transfer well to reality were noted and similar training data was generated which tended to improve transferability overall.

The collection of sufficient behavioural data was a time-consuming process and it was infeasible to get complete coverage of the search space. However, the development and tuning of a physics-based engine could become equally, if not more, time-consuming and would require specialized knowledge about the dynamics of the robotic system.

VII. CONCLUSIONS AND FUTURE WORK

For simple robots, SNNs have been shown to perform well as an alternative to physics-based approaches during the ER process. Research into using SNNs during the ER process for more complex robots is largely unexplored. The difficulty in developing adequate simulators increases as the complexity of the robots, environments and tasks increase. The SNN approach provides a simple way for researchers to construct models of reality that can be used to evolve behaviours that would be too costly to develop through real-world evaluations.

This paper demonstrated that SNNs are indeed viable for evolving behaviours for snake-like robots during the ER process. It was observed that certain locomotion modes were more stable and transferable than others. It is significant that this research was able to develop navigational controllers in simulation using a GA without the need for developing a physics-based simulation.

Future work could include, training separate SNNs for each locomotion mode could provide more accurate results. Future research could also include investigating the performance of various ANN morphologies, alternative training data sampling strategies and exploring a wider variety of tasks and robots.

ACKNOWLEDGMENT

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged (UID number: 89526). Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

REFERENCES

[1] D. K. Pratihar, "Evolutionary robotics - A review," *Sadhana*, vol. 28, no. 6, pp. 999–1009, 2003.

[2] I. Harvey, P. Husbands, D. Cliff, and Others, *Issues in evolutionary robotics*. School of Cognitive and Computing Sciences, University of Sussex, 1992.

[3] C. J. Pretorius, M. C. du Plessis, and C. B. Cilliers, "Simulating robots without conventional physics: A neural network approach," *Journal of Intelligent & Robotic Systems*, vol. 71, no. 3-4, pp. 319–348, 2013.

[4] C. J. Pretorius, M. C. du Plessis, and J. W. Gonsalves, "A comparison of neural networks and physics models as motion simulators for simple robotic evolution," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 2793–2800.

[5] J. C. Zagal and J. Ruiz-Del-Solar, "Combining simulation and reality in evolutionary robotics," *Journal of Intelligent and Robotic Systems*, vol. 50, no. 1, pp. 19–39, Mar. 2007.

[6] C. J. Pretorius, M. C. du Plessis, and C. B. Cilliers, "Towards an artificial neural network-based simulator for behavioural evolution in evolutionary robotics," in *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*. ACM, 2009, pp. 170–178.

[7] C. J. Pretorius, "Artificial Neural Networks as simulators for behavioural evolution in evolutionary robotics," Masters thesis, Nelson Mandela Metropolitan University, 2010.

[8] S. Hasanzadeh and A. Akbarzadeh, "Development of a new spinning gait for a planar snake robot using central pattern generators," *Intelligent Service Robotics*, vol. 6, no. 2, pp. 109–120, 2013.

[9] S. Hasanzadeh and A. A. Tootoonchi, "Ground adaptive and optimized locomotion of snake robot moving with a novel gait," *Autonomous Robots*, vol. 28, no. 4, pp. 457–470, 2010.

[10] K. Inoue, S. Ma, and C. Jin, "Optimizationof CPG-network for decentralized control of a snake-like robot," in *Robotics and Biomimetics (ROBIO). 2005 IEEE International Conference on*. IEEE, 2005, pp. 730–735.

[11] J.-K. Ryu, N. Y. Chong, B. J. You, and H. I. Christensen, "Locomotion of snake-like robots using adaptive neural oscillators," *Intelligent Service Robotics*, vol. 3, no. 1, pp. 1–10, 2010.

[12] T. Kamegawa, F. Matsuno, and R. Chatterjee, "Proposition of twisting mode of locomotion and ga based motion planning for transition of locomotion modes of 3-dimensional snake-like robot," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2. IEEE, 2002, pp. 1507–1512.

[13] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji, "Automatic locomotion design and experiments for a modular robotic system," *Mechatronics, IEEE/ASME Transactions on*, vol. 10, no. 3, pp. 314–325, 2005.

[14] S. Murata and H. Kurokawa, "Self-reconfigurable robots," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 1, pp. 71–78, 2007.

[15] H. H. Lund and O. Miglino, "From simulated to real robots," in *Evolutionary Computation, Proceedings of IEEE International Conference*. IEEE, 1996, pp. 362–365.

[16] J. C. Bongard, *Evolutionary robotics*. ACM, 2013, vol. 56, no. 8.

[17] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in artificial life*. Springer, 1995, pp. 704–720.

[18] D. Floreano, P. Husbands, and S. Nolfi, "Evolutionary robotics," *Springer handbook of robotics*, pp. 1423–1451, 2008.

[19] O. Miglino, K. Nafasi, and C. Taylor, "Selection for wandering behavior in a small robot," *Artificial Life*, vol. 2, no. 1, pp. 101–116, 1994.

[20] S. Nakamura, R. Saegusa, and S. Hashimoto, "A hybrid learning strategy for real hardware of swing-up pendulum," *Journal of Advanced Computational Intelligence & Intelligent Informatics (JACIII)*, vol. 11, no. 8, 2007.

[21] K. Melo, M. Hernandez, and D. Gonzalez, "Parameterized space conditions for the definition of locomotion modes in modular snake robots," in *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2032–2038.

[22] K. Melo, L. Paez, and C. Parra, "Indoor and outdoor parametrized gait execution with modular snake robots," in *2012 IEEE International Conference on Robotics and Automation*.

[23] K. J. Dowling, "Limbless locomotion: learning to crawl with a snake robot," Ph.D. dissertation, NASA, 1996.

[24] O. Shmakov, "Snakelike robots locomotions control," *Mechatronics–Foundations and Applications*, 2006.

[25] Itseez, "Official OpenCV Site," http://opencv.org, accessed: July 2015.

[26] Heaton Research, "Official Encog Site," http://www.heatonresearch.com/encog, accessed: July 2015.

Appendix B

Robotics and Autonomous Systems

An initial prototype of the proposed BNS approach was developed and investigated for the Khepera robot in this study. The viability of the BNS approach was successfully validated by concurrently developing SNNs and controllers for a real-world Khepera robot. Controllers were developed for trajectory planning tasks. An extensive parameter comparison study of the proposed approach was conducted in order to identify and study influential factors.

This study was accepted and published in the Robotics and Autonomous Systems (affiliated with the Intelligent Autonomous Systems (IAS) Society) journal. The paper follows.

Concurrent Controller and Simulator Neural Network Development for a Differentially-Steered Robot in Evolutionary Robotics

Grant W. Woodford[a], Christiaan J. Pretorius[b], Mathys C. du Plessis[a]

[a]*Department of Computing Sciences, Nelson Mandela Metropolitan University (NMMU), Port Elizabeth, South Africa*
[b]*Department of Mathematics and Applied Mathematics, Nelson Mandela Metropolitan University (NMMU), Port Elizabeth, South Africa*

Abstract

Evolutionary Robotics (ER) strives for the automatic creation of robotic controllers and morphologies. The ER process is normally performed in simulation in order to reduce the time required and robot wear. Simulator development is a time consuming process which requires expert knowledge and must traditionally be completed before the ER process can commence. Traditional simulators have limited accuracy, can be computationally expensive and typically do not account for minor operational differences between physical robots.

This research proposes the automatic creation of simulators concurrently with the normal ER process. The simulator is derived from an Artificial Neural Network (ANN) to remove the need for formulating an analytical model for the robot. The ANN simulator is improved concurrently with the ER process through real-world controller evaluations which continuously generate behavioural data. Simultaneously, the ER process is informed by the improving simulator to evolve better controllers which are periodically evaluated in the real-world. Hence, the concurrent processes provide further targeted behavioural data for simulator improvement.

The concurrent and real-time creation of both controllers and ANN-based simulators is successfully demonstrated for a differentially-steered mobile

Email addresses: grant.woodford@nmmu.ac.za (Grant W. Woodford), cpretorius@nmmu.ac.za (Christiaan J. Pretorius), mc.duplessis@nmmu.ac.za (Mathys C. du Plessis)

robot. Various parameter settings in the proposed algorithm are investigated to determine factors pertinent to the success of the proposed approach.

Keywords:
Evolutionary Robotics, Coevolution, Simulator, Artificial Neural Networks, Differentially-Steered, Mobile Robotics

---

1. INTRODUCTION

Evolutionary Robotics (ER) is a field of study that involves the automatic artificial evolution and optimization of particular traits of autonomous robotic systems [1]. ER seeks to automate the development of robot controllers and morphologies through the use of Evolutionary Computation as an alternative to their manual creation. As robots, environments and tasks become more complex, the manual creation of controllers becomes more infeasible [2]. ER can be used to evolve robot behaviours such at path following, inverted pendulum stabilization, light following, obstacle avoidance and many others [3, 4]. In ER, controllers need to be evaluated to quantify the relative performance of each controller at performing a given task. ER requires the evaluation of a large number of controllers in order to evolve better ones. However, evaluating a large number of controllers on a real-world robot is unrealistically time-consuming and can damage hardware through mechanical wear [5]. To overcome these issues, simulators serve as an alternative to reality for evaluating controller performance in the ER process [5].

There are many different types of simulators. Simulators can be broadly classified into three categories, namely physics based simulators, empirical models (based on experimentally collected data) and hybrids which are a combination of both physics and empirical models [3]. Traditional methods for creating simulators are often time-consuming and complicated because it may require the construction and tuning of complex physics based models or the collection of large amounts of real-world data or both.

Much research in ER is concerned with overcoming the challenges in using simulators effectively [1]. Challenges in simulator design are inaccuracies and/or over-simplification in the modelling of certain phenomena. Oversimplified or inaccurate simulators may result in controllers that rely too heavily on peculiarities that exist only in simulation but are not present when the controller is evaluated in reality, commonly referred to as the *reality gap*

problem [6]. Oversimplification can be avoided by using highly accurate simulators. However, even highly accurate simulators cannot model reality perfectly and will inevitably contain inaccuracies [7]. Highly accurate simulators can also be computationally expensive [8]. Scalability can become an issue where the time taken to evaluate controllers can grow substantially with increased complexity in the robotic system [1, 9]. Thus, simulators ideally need to provide highly accurate representations of reality whilst not being too computationally expensive to operate.

Pretorius, du Plessis and Cilliers [10] have shown that Artificial Neural Networks (ANN) can be utilized as simulators in the ER process. A simulator created using ANNs shall be referred to as a Simulator Neural Network (SNN). SNNs have been shown to possess good prediction accuracy, noise-tolerance and generalization abilities in the modelling of certain robot behaviours [3]. SNNs are computationally efficient and can be created without the need for specialized knowledge about the dynamics of the robotic system. It has also been shown that SNNs can aid in the successful transference of controller behaviour from simulation to reality [3].

Previous approaches to SNN construction required the gathering of large quantities of data from real-world robot behaviour which can be used to train SNNs before the ER process begins. This development of simulators before controllers are evolved is time-consuming. The robot generates the behavioural data by evaluating randomly generated commands. However, if only certain behaviours need to be modelled then much of this behavioural data is unnecessary. A simulator could specialize in accurately modelling only the behaviour required to perform a given task and thus require mostly behavioural data specific to the desired behaviour.

As an alternative to the traditional approach to SNN creation, the current work aims to investigate the concurrent creation of SNNs and controllers in the ER process. This approach could potentially speed up the process by eliminating the need to pre-compute SNNs. The SNNs would be specialized to accurately model only behaviours required for a given task and therefore require less empirical data for their development than the previous approach. There could also potentially be further advantages warranting investigation such as the ability to adapt to changing environments and robots. Automatic damage recovery capabilities could also be possible since the simulator is no longer static.

The related work (Section 2) gives an outline of some of the work previously done in this area. The proposed approach of concurrent simulator and

controller development is outlined in Section 3. The evaluation methodology and experimental procedure used for investigating the proposed approach are then discussed in Sections 4 and 5, respectively. The results of the experimental work are presented (Section 6) and finally conclusions are drawn and future work is suggested (Section 7).

2. RELATED WORK

This section reviews important approaches in the development of simulators and controllers for the ER process. Section 2.1 discusses related work with regards to concurrent controller and simulator development. Section 2.2 discusses the development and use of SNNs in ER.

2.1. Concurrent Controller and Simulator Development

One approach to dealing with the *reality gap* problem described in Section 1 is to evolve controllers on real-world hardware only [11]. On the other hand, a combined approach can be taken where controllers are evolved in simulation then transferred to a real-world robot where the ER process continues in reality [12].

The one directional transference of a controller from simulation to reality can also be replaced by a more bidirectional approach [1]. There have been many proposals for bidirectional approaches that allows the optimization process to alternate between the simulator and real-world [5, 13–15].

A robot's behaviour is affected not only by its controller but also by the robot's morphology. A coevolutionary approach can be taken where controllers and robot morphologies are improved together [16, 17]. Evolving a robot's morphology usually necessitates changing the model of what the robot looks like and evolving the physics model parameters of the simulator [14, 18]. If a robot's model is fixed, then only the physics model parameters of the simulator could be evolved. Work related to coevolutionary approaches to controller and simulator development are thus important.

One example of the coevolution of simulator model parameters and controllers is the Anytime Learning Algorithm [13]. Anytime learning proposes a population of model parameters and a population of controllers. These populations are evolved by means of a Genetic Algorithm (GA). A population of model parameters are maintained and judged against real-world data to determine their accuracy. The most accurate model parameters are used for controller evolution.

Bongard, Zykov and Lipson [14] proposed the Estimation-Exploration Algorithm (EEA) that integrates robotic self modelling into the ER process. The EEA is a hybrid coevolutionary algorithm for maintaining populations of models of the robot system and a population of test controllers used to explore the model search space. Once a sufficiently accurate model has been found, it is used to evolve controllers to achieve the required behaviour.

The Transferability Approach has also been proposed [15]. This method does not attempt to evolve the simulator but rather attempts to complement it. Simulators often have limitations in the accurate modelling of certain phenomena in order to increase computational performance [15]. If these limitations are known during the ER process, controllers could avoid relying on dynamics that have not been accurately simulated. The Transferability Approach proposes a multi-objective fitness function of two parts. One part estimates how well a solution may transfer from simulation to reality (called the transferability function) and another estimates how well the desired behaviour is achieved in simulation [15]. The transferability function can be implemented as an ANN or Support Vector Machine and may be trained during the ER process. Controllers evaluated on a real-world robot generate data that is utilized in the training of the transferability function. Behavioural features that can be measured in simulation and reality are identified and the transferability function is trained to estimate how well these behaviours are simulated.

The Back to Reality (BTR) Algorithm concurrently evolves controllers and simulators [5]. The BTR Algorithm collects the fitness of controllers evaluated in simulation and reality. The fitness function for evolving simulators is taken as the average difference between controller fitness in simulation and reality. The optimization process tries to minimize this value. The BTR Algorithm has shown success evolving behaviour using a realistic dynamics simulator [5, 19].

A Genetic Programming (GP) approach has shown success for the creation and tuning of a physics based simulator for modelling the dynamics of a rotorcraft [20]. A non-linear model was developed using GP and real-world data generated from test controllers, after which the final model was used to develop controllers for specific tasks.

Many of these bidirectional approaches require the initial development of a simulator before controllers can be evolved. The initial construction of these simulators may require much human intervention after which only certain aspects of the simulator are improved using bidirectional approaches.

The approach proposed in this paper (Section 3) used simple SNNs with zero knowledge about the dynamics of the robotic system and autonomously developed the simulator and controllers during the ER process.

2.2. Simulator Neural Networks

Researchers have proposed SNNs as an alternative to traditional simulators [10, 21, 22]. SNNs have been utilized as simulators in the ER process to create controllers for tasks such as obstacle avoidance, light approaching behaviour and inverted pendulum stabilization [3]. Research has also shown that SNNs can simulate the dynamics of the pendulum swing-up problem [21].

SNNs are trained using data experimentally collected from a real-world robotic system. The SNNs can then be used to determine the fitness of candidate controllers instead of running controllers on real-world hardware.

The conventional approach to creating a SNN and evolving controllers using the ER process is as follows [3]:

1. Randomly generated commands are performed on a real-world robot. As a result of these evaluations, the robot moves around the environment and data is collected using motion tracking and sensor logging techniques.
2. When sufficient data has been collected, it is used to train SNNs to predict the real-world robot's motor and sensor behaviour.
3. The trained SNNs are used to evaluate the fitness of candidate controllers during controller evolution.
4. At the end of the evolutionary process, the best controller is transferred and evaluated using the real-world robot.

Some advantages of SNNs are that they possess good generalization abilities, prediction accuracy and can handle large quantities of noise [3]. Physics models require explicit mathematical models whereas SNNs do not. The construction of a SNN can be simpler than that of a physics based simulator because no prior knowledge is required of the physics governing the robotic systems. It has also been shown that SNNs can be computationally more efficient, more accurate and result in greater transferability to reality when compared to physics-based models [4].

As was previously mentioned, there are disadvantages to developing simulators before controllers. The simulator must be created before the ER

process can be initiated. The simulator may also be trained to simulate real-world behaviours that are often not required for the successful evaluation of controllers for the goal task. As a consequence to these disadvantages, many time consuming real-world evaluations may be needed to create a simulator. These SNNs remain static and thus are unable to take into account changes due to mechanical wear or environmental factors. Motor speeds may be heavily influenced by the battery level of the robot [10]. If the robot's battery level fluctuates, static SNNs are unable to take into account battery levels, unless specifically trained to.

3. PROPOSED APPROACH

Bidirectional approaches to simulator development have shown much promise (Section 2.1). Pre-computed SNNs have also shown much potential as robotic simulators (Section 2.2). This paper thus proposes a bidirectional approach to SNN and controller development. The approach involves a bootstrapping process whereby controllers are continually evaluated on a real-world robot, thereby generating training data for the improvement of the SNNs. The improved SNNs are used to evolve better controllers and these controllers can then be evaluated on the robot to provide further training data to the SNNs. This process improves the SNNs and controllers concurrently so that controllers evaluated in reality would ideally converge towards the desired behaviour. Consequently the SNNs would more accurately model the desired behaviour and be less able to model unnecessary behaviour. Such an approach can speed up the ER process by eliminating the need to pre-compute SNNs before controllers are evolved and may require fewer real-world evaluations than pre-computed SNNs.

The proposed approach is illustrated in Figure 1 and consists of the following steps:

1. Controllers are selected for real-world evaluations from the population of controllers. Behavioural data from the real-world evaluations are collected and stored in a training data buffer.
2. The training data stored in the training data buffer periodically integrates into SNN training to better predict robot behaviours.
3. The training SNNs are periodically copied and used to replace the controller evolution SNNs. The population of controllers are evolved using the controller evolution SNNs to estimate controller fitness.

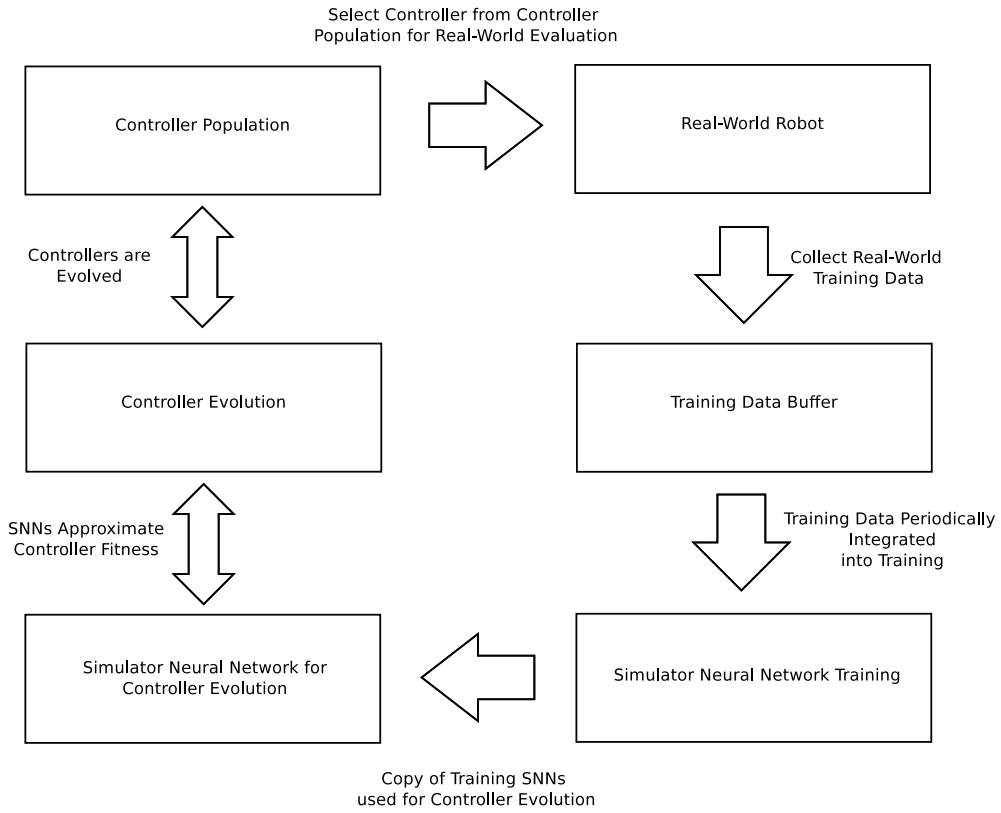


Figure 1: The proposed approach

4. The process continually repeats by going back to step 1 or may terminate when stopping conditions are met.

The following section discusses the evaluation methodology used for demonstrating the viability of this proposed approach. The methodology used to identify influential factors related to its success or failure is also discussed.

4. EVALUATION METHODOLOGY

The viability of the proposed approach mentioned in the previous section was demonstrated on a differentially-steered robot (Section 5.1) to perform three different trajectory planning tasks of varying levels of complexity. ER

was used to evolve waypoint navigational controllers (Section 5.2) in simulation while simultaneously the simulator was optimised using resilient backpropagation training (Section 5.3). A controller was periodically selected from the population of open-loop array-based controllers and evaluated on a real-world robot. These evaluations were tracked using motion tracking techniques and the behavioural data generated was collected and used for training the simulator. The simulator used in the controller evolution process would thus commence untrained and would gradually improve as more behavioural data was collected. Section 5.4 describes how the proposed approach was investigated on the real-world robot and results are presented in Section 6.1.

The behaviour of the proposed approach for various parameter settings was subsequently investigated. It was anticipated that the diversity of the controller population was a key factor because it governed the controller search space explored during the ER process and also influenced simulator training. The analysis was thus focused on how the parameters affected the diversity and how the diversity in turn manifested into the success of the controllers. The diversity in relation to the effectiveness of the simulator to predict robot behaviour was also analysed. This experimental investigation required the testing of a large number of parameter combinations which was made possible by using an already trained simulator as an alternative to the real-world robot. The substitute real-world robot expedited the process and made the experiment viable. The parameter settings investigation was successful in determining influential factors related to the success or failure of the proposed approach. Section 5.5 describes how parameters were investigated while Section 6.2 discusses the results. The following section provides detailed information regarding the experimental procedure.

5. EXPERIMENTAL PROCEDURE

The following sections provide details regarding the experimental work and procedures that were followed. Section 5.1 discusses the hardware and data capturing techniques. Details of the controller used are discussed in Section 5.2 and the developed simulator in Section 5.3. The real-world prototype and parameter comparison experiments are lastly addressed in Sections 5.4 and 5.5, respectively.

5.1. Hardware and Data Capture

The robot used for the experimental work is the Khepera III mobile robot [23]. The reason this robot was chosen, is because of the robot's frequent use in ER [7, 11, 15, 24] and ease of use. The robot is differentially-steered and movement is controlled by two separately driven wheels. The Khepera is controlled by sending commands over a Bluetooth serial interface. Steering was accomplished by varying the relative rate of rotation of the wheels. The environment upon which the robot operated was a horizontal skid-proof surface. The Khepera was modified by mounting three LED infra-red lights on top. A Nintendo Wii remote was positioned approximately 1.8 meters above the robot to track the LEDs so that behavioural data could be captured [25, 26].

5.2. Controllers

The controllers developed for the Khepera were for waypoint navigation. A controller in this paper is defined as a list of commands that when executed on a robot would cause it to maneuver in a particular trajectory without feedback. Waypoint navigation planning was chosen due to its simplicity. Controllers therefore did not have access to the robot's real-time position during evaluation. Controllers consisted of a variable length sequential list of commands, each consisting of the left motor speed, right motor speed and the time duration for which the command must be executed. This was the first study into the proposed approach and in order to keep the investigation simple, motor speeds were restricted to moving forwards only. The proposed approach began by creating a population of randomly generated controllers. Between four and thirteen commands were generated for each controller in the initialized population but subsequent generations could create controllers that could contain an unlimited number of commands through crossovers. A controller was evaluated by executing the list of commands sequentially on the robot. Collisions with the boundaries of the working surface were not considered.

Controllers were developed for three different waypoint navigation tasks. Figures 3 to 5 demonstrate the paths followed by the robot for each task. The robot's starting position was located centrally amongst the goal points and faced northward. Each of the three tasks were described as a sequential list of goal points on the operating surface. Success was accomplished when the robot traversed all these points in the given order. The robot was judged to have reached a given goal point if it moved within seven centimetres of the

point. The seven centimetre leniency was chosen to allow for more leeway in reaching goal points during controller evolution. The first task, referred to as Task 1, was a circular path (Figure 3). The second task, referred to as Task 2, was an infinity sign path (Figure 4). The third task, referred to as Task 3, was a complex path (Figure 5).

The end result of a controller evaluation, was a list of positions reached by the robot at the end of each command. There would also be a list of goal points the robot needed to reach in the specified order. The pseudo-code for how controller fitness was calculated is shown in Algorithm 1. This algorithm took as input the list of positions and goal points. During the traversal of the positions list, the distance to the first goal point was calculated. Once that goal point was reached, the distance to the next goal point in the list was used instead and so on. For each of the positions in the list, the distance to its goal point was calculated. These distances are summed together and a penalty distance was added for each missed goal point. This total distance would ideally be minimized during controller evolution and the fitness was calculated as its inverse.

Algorithm 1: Controller fitness evaluation

```
Data:
positions ← List of positions reached by robot
goalpoints ← List of goal points
Result: Fitness of controller
penalty ← A large constant integer value
goalPointIndex ← 0
sum ← 0
for each position in positions do
    if position reaches goalpoints[goalPointIndex] then
        goalPointIndex ← goalPointIndex + 1
    end
    sum ← sum + (distance to goalpoints[goalPointIndex])
end
sum ← sum + penalty * (number of goal points not reached)
fitness ← 1/sum return fitness
```

The fitness function was designed to minimize the number of commands used to accomplish the task. Controllers were continually evolved throughout

the process using a GA. Elitism was used, with the best performing controller for each generation carrying over to the next.

5.3. The Simulator

The operating surface can be mathematically represented as an xy-plane of a Cartesian coordinate system, known as the global coordinate system. The robot's movements were captured based on the global coordinate system. The robot had its own local coordinate system where the origin was always the centre of the robot's wheel axis and the y-direction was aligned with the robot's forward heading. The simulator was trained to predict changes in relation to the robot's local coordinate system.

The simulator was used to assign fitness values to the population of controllers. In the beginning, the simulator was untrained, so fitness assignments would effectively be random. However, the quality of the fitness assignments gradually improved as more training data became available from real-world controller evaluations. A training pattern consisting of the previous left motor speed, previous right motor speed, current left motor speed, current right motor speed, time duration of execution, the robot's local x-displacement, the robot's local y-displacement and change in the robot's orientation angle. Captured training patterns were added to either the training data set or verification data set. There was a 75% probability of data being added to the training data set and a 25% probability of being added to the validation set. The validation set was not utilized during simulator training, but was used to analyse potential issues in over-fitting and accuracy. It was found that over-fitting did not occur during training.

The simulator consisted of three separate SNNs, each consisting of a single Feed Forward Neural Network (FFNN) (Figure 2). The FFNN setup was chosen based on previous investigations. The simulator was separated across multiple FFNNs to produce more accurate results [10]. Each FFNN had 5 input neurons, 20 hidden neurons and 1 output neuron. An indication of the optimal number of hidden neurons was determined experimentally during previous studies [27]. The input and output neurons made use of a linear activation function while the hidden neurons made use of a sigmoid activation function.

All FFNNs took as input the previous left motor speed, previous right motor speed, current left motor speed, current right motor speed and the time duration of the current command. The outputs of each of the separate

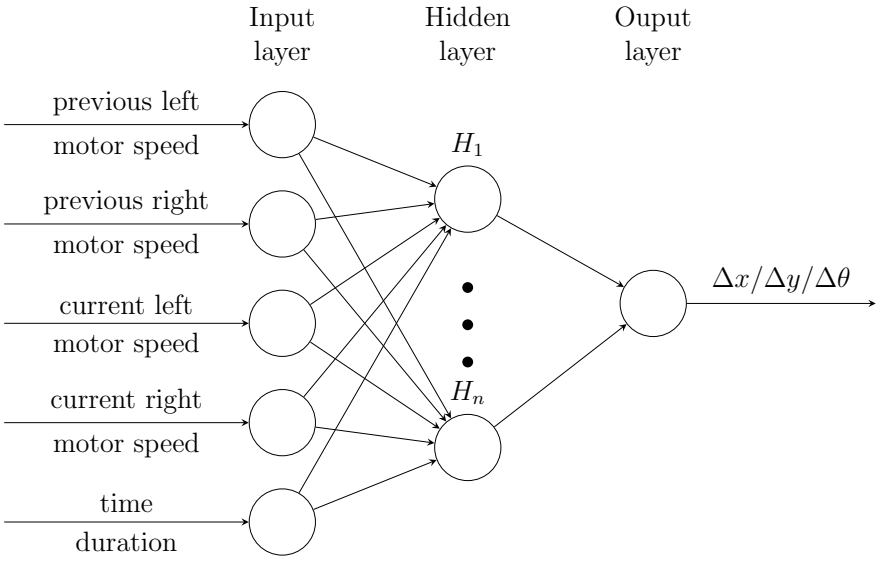


Figure 2: Simulator Neural Networks

FFNNs were the x-displacement (Δx), y-displacement (Δy) and change in angle ($\Delta\theta$).

SNNs were implemented using an open source machine learning library called Encog [28] and trained used the Resilient Backpropagation algorithm [29]. There were two types of SNNs, the training SNNs and the controller evolution SNNs. A copy of the training SNNs was periodically made and used for the controller fitness assessments (referred to as the controller evolution SNNs). The training SNNs were periodically trained for 1000 iterations using Resilient Backpropagation, after which new training data was added from the training data buffer.

To ensure that controllers were better able to cross the *reality gap*, noise could be added to the simulator during controller evaluations [6]. However, noise was not added to the controller evolution SNNs for any of the experiments presented in this investigation. The reason noise was not utilized during controller evolution was because it was observed that controllers were able to transfer well to reality without the addition of noise.

5.4. Prototype Experiments

The proposed approach was performed on the real-world hardware mentioned in Section 5.1. Experiments proceeded until the robot was visually perceived to have consistently traversed all the goal points accurately. Each of the three waypoint navigation tasks were completed using the proposed approach over three trials.

The parameters used for controller evolution are given in Table 1. Optimal parameter values were determined experimentally based on the parameter comparison experiments described in Section 5.5.

Table 1: Parameters for controller evolution

Controller Population Size	400
Initialization	Random from a uniform distribution
Selection	Tournament (Tournament size 50%)
Crossover Method	Two parent
Mutation Probability	10%
Mutation Method	Random Component Perturbation

The Controller Population Size parameter (Table 1) was the size of the population of controllers used for controller evolution. The initial population of controllers were created such that each controller contained a random number of commands, each command being generated randomly from a uniform distribution. The range of motor speeds were based on the operational limits of the robot and were between 3000 and 30000 and the duration of each command was between 400 and 3000 milliseconds. Durations below the lower limit would be difficult to measure and predict while durations above the upper limit could result in rotation angles that were larger than 180 degrees.

The selection operator used for controller crossover was tournament selection [30]. A Tournament Selection Size of 50% of the controller population was chosen and a two parent crossover method was used for controller evolution. Two parent crossover involved two parent controllers, for which the command sequence sizes of both parents may be of differing lengths. A random crossover point was chosen for each parent which resulted in each parent

forming two segments of commands. The child controllers were created by interchanging the first segments of each parent. There was an 80% probability that the crossover points of two parents were identical and a 20% probability that they could be different. This crossover method was chosen to allow for the number of commands to be optimized while not unnecessarily distorting existing solutions. The Mutation Rate was the probability that the left motor speed, right motor speed or duration of a command may change by a random amount. The Mutation Rate chosen for controller evolution was 10%.

An additional parameter which was specific to the proposed method was the Real-world Selection parameter. At the end of every real-world controller evaluation a new controller was selected for the next real-world evaluation. This new controller was selected from the population of controllers using tournament selection and the tournament size was specified by the Real-world Selection parameter. A Real-world Selection Size of 70% of the controller population was chosen for the experimental work.

5.5. Parameter Comparisons

In order to investigate important factors that contributed to the success or failure of the proposed approach, the influence various parameter settings had on success was investigated. The parameters tested are listed in Table 2. Due to the stochastic nature of the experiments, every possible combination of the parameter settings was independently run for 30 trials for each task.

Table 2: Parameter values used for comparisons

Controller Population Sizes	50, 100, 200, 400
Tournament Selection Sizes	10%, 20%, 30%, 40%, 50%
Mutation Probability Rates	10%, 30%, 50%, 70%, 90%
Real-world Selection Sizes	10%, 30%, 50%, 70%, 90%

The investigation of such a large number of parameter combinations on a real-world robot would take infeasibly long due to the time required to run them. A pre-computed simulator based on previous studies was thus used as a substitute for the real-world robot [27], referred to as the static simulator. Training data generated from the real-world robot would inevitably contain

errors due to inconsistencies in motor function and sensor readings [27]. To accurately simulate reality, noise was thus added to the static simulator in order to realistically replicate noise that would be present. The noise distribution was assumed to be Gaussian with mean zero and with variances based on the difference between the static simulator and real-world data. The static simulator with noise which acted as a substitute to the real-world robot will be referred to simply as the substitute real-world.

The substitute real-world also simulated the actual time required to evaluate controllers in reality. This artificial slowing down of the substitute real-world was done to allow for the SNN training and controller evolution to progress with durations similar to reality. Each experimental run was progressed until the 100^{th} controller was evaluated in the substitute real-world. Achieving the desired behaviour beyond the hundredth evaluation was considered impractical for real-world experiments.

A controller successfully completed a given task if during evaluation the robot passed through all goal points in the assigned order. Goal points were defined as points in the xy-plane forming a circled region with a radius of seven centimetres. Success rates over time were calculated for varying goal sizes for every combination of parameter settings. The varying goal point sizes had a radius of either twenty, fourteen, ten or seven centimetres. These various levels of success rate were used to identify how close the substitute real-world robot was able to achieve the desired behaviour.

The best fitness value during controller evolution was periodically recorded and the average fitness of each parameter combination over the 30 trial runs was calculated. The success rate for a given parameter combination was its success ratio out of the 30 trial runs. The success rates for the various goal point sizes, along with the average fitness of controllers evaluated for each parameter combination was used to rank how well they performed overall.

Ranking how well the various parameter combinations performed was done by ordering the success rates according to the seven centimetre goal point radius, then ten, fourteen, twenty and lastly on the average fitness. The top ten best and worst ten parameter combinations could thus be identified through this ranking method. The reason for not simply using average fitnesses for ranking was because the proposed approach could sometimes fail if controller evolution converged towards sub-optimal solutions. Experiments that resulted in a sub-optimal solution would thus result in poor fitness values. These poor fitness values often varied greatly in value and could unfairly skew the average fitness of a parameter combination. A ranking based on or-

dering parameter combinations according to how well the desired behaviour was achieved was thus chosen.

The diversity of the controller population during controller evolution was measured. The relationship between the diversity and various other properties such as parameter settings, controller success rates and how well the developed simulator was able to predict the substitute real-world fitnesses were also studied. The diversity for a given controller population was calculated using Equation 1. The motor speeds and time durations for all controllers were normalized to ensure that there was equal weighting of importance. The normalized left motor speed, right motor speed and time duration of the j^{th} command for the i^{th} controller in the population of controllers are denoted by L_{ij}, R_{ij} and T_{ij} respectively. The controller population size is denoted by n and the total number of commands for the i^{th} controller is denoted by r_i. The average normalized left motor speed, right motor speed and time duration for the j^{th} command over all controllers is denoted by $\overline{L}_j$, $\overline{R}_j$, and $\overline{T}_j$ respectively.

$$\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{r_i}|L_{ij}-\overline{L}_j|+|R_{ij}-\overline{R}_j|+|T_{ij}-\overline{T}_j|) \quad (1)$$

An obvious requirement for the proposed approach to perform well was the need for controller fitness estimates in simulation to be fairly close to the substitute reality. A fitness error was calculated by computing the absolute difference between a controller's fitness in simulation versus the substitute real-world (in other words a measure of how well the developed SNNs simulated the substitute real-world). At the end of each substitute real-world evaluation, data was collected regarding the controller population diversity, differences between the fitness in simulation and substitute real-world (referred to as the fitness error) and the success at varying levels of accuracy.

In order to determine how well controllers perform over the lifetime of an experimental run, the relationship between the developing simulator's perceived fitnesses and the substitute real-world fitnesses over time were studied. A single experimental run of the proposed approach was conducted using the substitute real-world robot. The parameter settings chosen for the experiment were identical to the real-world validation experiments. During the ER process, the best controller's fitness in simulation was recorded for every generation. The best controller's corresponding substitute real-world fitness was also calculated for every generation. The change between subsequent

simulators used for the controller evolution process was also studied.

6. RESULTS AND DISCUSSION

Section 6.1 discusses the results of the prototype experiments referred to in Section 5.4 and Section 6.2 discusses the results of the parameter comparison experiments described in Section 5.5.

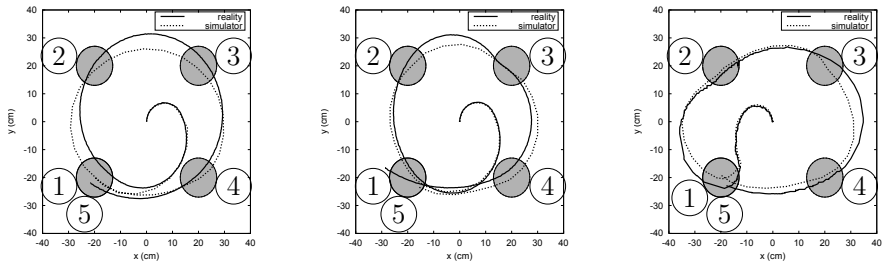
6.1. Prototype Results

The real-world prototype experiments were able to demonstrate that the proposed approach is indeed viable. The developed SNNs made sufficiently accurate predictions of robot behaviour. This allowed controllers to successfully evolve and demonstrate the required behaviours.

The three waypoint navigation tasks are presented in Figures 3 to 5. These figures represent the final controllers found for each trial run using the proposed approach. Real-world generations represent the number of controllers that were tested on the robot. Each experiment was run until the robot was visually perceived to have successfully and consistently performed the goal task, which was why the total number of real-world generations for the trials differ. In each figure, the solid line represents the real-world path followed by the robot when the fittest controller was evaluated. The dotted line represents the developed simulator's perception of how the robot would behave when the same controller was evaluated in reality. The circled grey regions are the goal points with a radius of 7cm. The order in which the goal regions must be traversed by the robot are annotated with circled numbers. The total run-time and number of real-world evaluations for each trial is given in Table 3. It can easily be seen that the prototype run-times do not seem to be consistent and do vary considerably.

As previously mentioned, controllers were developed for trajectory planning. Therefore, controllers were not aware of the position of the robot during evaluations which can result in an accumulation of errors. However, the prototype experiments showed that the trained SNNs were able to predict robot behaviour with enough accuracy to develop viable controllers.

Task 1 (Figure 3) had four goal points positioned 40cm apart from each other in a square formation. The first and second trial runs for Task 1 (Figures 3a and 3b) had similar solutions, where the robot made an initial right turn and produced a spiral shape. The third trial made use of an

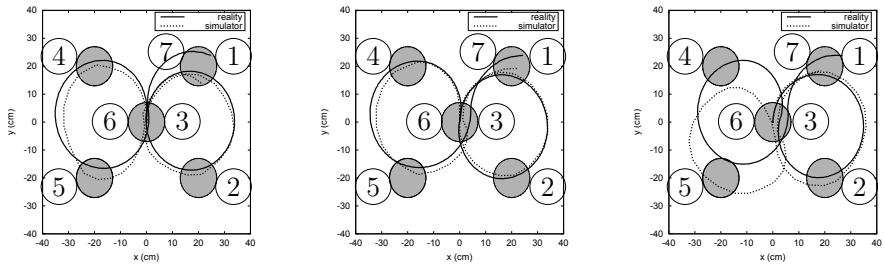


(a) First trial run at the 45th real-world generation

(b) Second trial run at the 20th real-world generation

(c) Third trial run at the 15th real-world generation

Figure 3: Task 1 prototype trial runs



(a) First trial run at the 40th real-world generation

(b) Second trial run at the 30th real-world generation

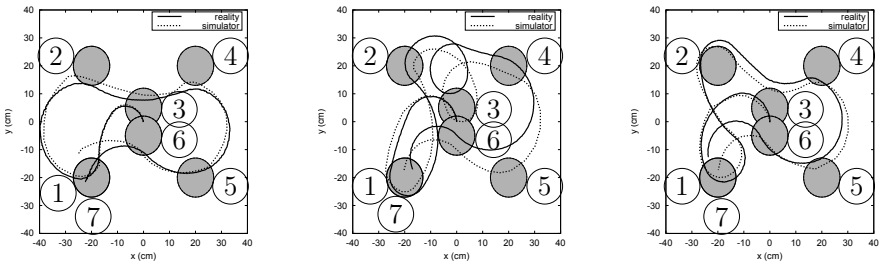
(c) Third trial run at the 20th real-world generation

Figure 4: Task 2 prototype trial runs

alternative strategy which differed from the first two trials by initially turning left instead.

Task 2 (Figure 4) had five uniquely positioned goal points where the centre one had to be reached twice by the robot. All solutions to the second task made use of the same strategy. The robot started at the centre position and traversed the goal regions in the annotated order. The real-world paths for the first two trials closely matched the simulators perception of reality (Figures 4a and 4b). The third trial run successfully performed the task in reality, however, the simulator's perception of reality was less accurate (Figure 4c).

Task 3 (Figure 5) contained six independent goal points and good solu-



(a) First trial run at the 55th real-world generation

(b) Second trial run at the 60th real-world generation

(c) Third trial run at the 45th real-world generation

Figure 5: Task 3 prototype trial runs

Table 3: Prototype run-times

Goal task	**Trial number**	**Run-times**	**Real-world evaluations**
Task 1	Trial 1	25 minutes	45
	Trial 2	12 minutes	20
	Trial 3	9 minutes	15
Task 2	Trial 1	25 minutes	40
	Trial 2	19 minutes	30
	Trial 3	12 minutes	20
Task 3	Trial 1	40 minutes	55
	Trial 2	50 minutes	60
	Trial 3	30 minutes	45

tions needed to perform many sharp and quick turns. Each trial solved Task 3 using a different strategy. The first trial contained no loops at all while the second and third trials contained two and one loops respectively. The second and third trials contained a loop around the bottom left goal point. The second trial produced a loop around the top centre goal point.

During the proposed approach, controllers evaluated in reality would converge towards the desired behaviour. However, there were instances where this convergence was lost, but gradually improved towards the desired behaviour again. This sudden loss of successful controllers was possibly due to the simulator periodically changing. Thus causing controllers that barely reached certain goal points to failed to reach them when the simulator was updated.

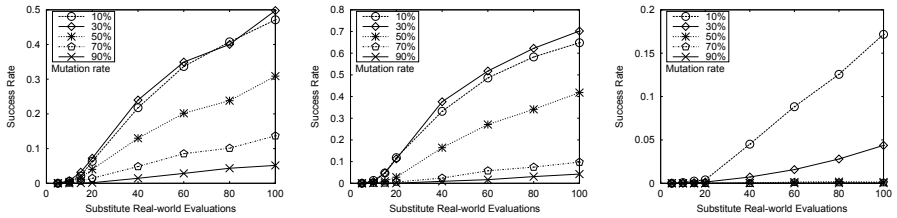
6.2. Parameter Comparison Results

The results of the parameter combination experiments described in Section 5.5 are discussed in this section. These experiments were conducted completely in simulation using a simulator (substitute real-world) as an alternative to a real-world robot. Section 6.2.1 discusses results related to all the parameter combination experiments, whereas Section 6.2.2 discusses results related to the best and worst performing parameter combinations.

6.2.1. Overall Parameter Combination Results

The success rate over time for each of the parameter values listed in Table 2 were determined over all parameter combination experiments. These success rates were used to determine which parameters were most influential in the success of the proposed approach.

The success rates of the tested controller mutation rates over all experiments is shown in Figure 6. The controller mutation rate was seen to be the most influential parameter tested. The largest difference between the mutation success rates was approximately 45% for Task 1, 66% for Task 2 and 17% for Task 3. Lower mutation rates generally performed better. Tasks 1 and 2 mostly performed better with a 30% mutation rate whereas Task 3 performed better with a 10% mutation rate.

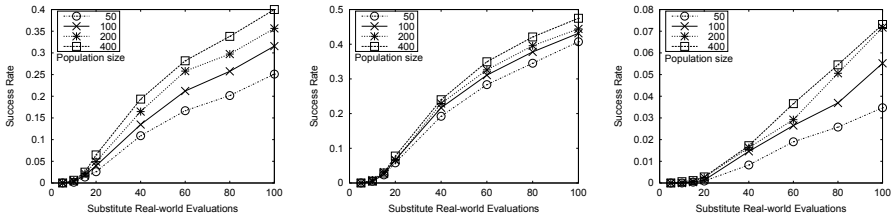


(a) Task 1 (b) Task 2 (c) Task 3

Figure 6: Success rate versus number substitute real-world evaluations for the tested controller mutation rates

The success rates over time for the tested controller population sizes over all experiments is shown in Figure 7. The controller population size was the second most influential parameter tested. The largest difference between the population size success rates was approximately 15% for Task 1, 7% for Task 2 and 4% for Task 3. It was found that larger controller population sizes

performed better. During controller evolution, good solutions may be lost due to a changing simulator or a high controller mutation rate. However, a higher population size may contribute towards reducing these issues.



(a) Task 1 (b) Task 2 (c) Task 3

Figure 7: Success rate versus number substitute real-world evaluations for the tested controller population sizes

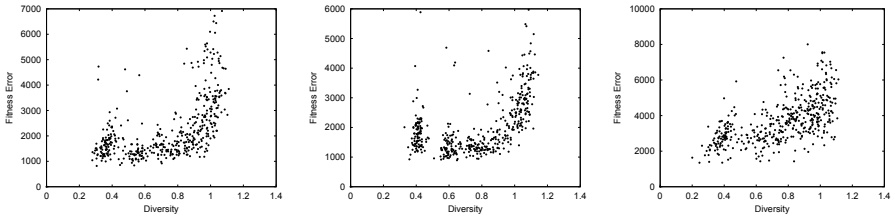
Success rates for the controller and real-world tournament selection sizes were not shown graphically due to their lower importance. The overall result showed that the controller tournament selection size was the third most influential parameter tested and generally performed better for larger values. The 10% controller tournament selection size performed the worst for all tasks while sizes 20% and above performed similarly better. The largest difference between controller tournament selection size success rates were 6.3% for Task 1, 6.7% for Task 2 and 1% for Task 3. The least influential parameter tested was the real-world tournament selection size which did not appear to have any discernible trend across the different tasks. The largest difference between the success rates of the real-world tournament sizes were 3% for Task 1, 2.5% for Task 2 and 1.7% for Task 3.

In order to determine how well the simulators performed in relation to diversity, the fitness error was measured. This measures the difference between a controller's fitness in the developed simulator and substitute real-world. The average fitness error versus population diversity for each parameter combination after the 100^{th} substitute real-world evaluation can be seen in Figure 8. The success rate versus diversity of the tested parameter combinations were also investigated. The success rates versus diversity for each parameter combination is shown in Figure 9.

Figures 8 and 9 indicate that a low diversity had a wider range of fitness errors and success rates when compared to the optimum diversity range.

A low diversity may indicate the premature convergence of the controller population. This convergence to suboptimal solutions allowed the simulator to accurately model the behaviour, however, tasks were never successfully completed.

A very high diversity may indicate that controllers were continually performing an exploratory search of the search space and thus resulted in controllers being unable to exploit good solutions. Figure 8 shows that the average fitness error generally increased after the diversity passed a certain point. The increase in the fitness error may have been a result of controllers evaluated in the substitute real-world not converging towards the desired behaviour. The controllers would thus be less specialized and the training data for the simulator would be less targeted towards the desired behaviour which made simulator training more challenging.

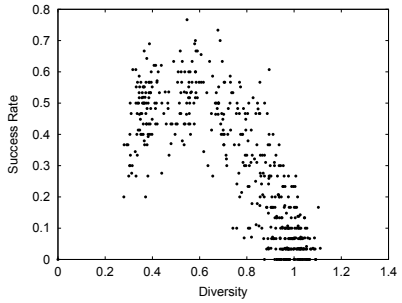


(a) Task 1 (b) Task 2 (c) Task 3

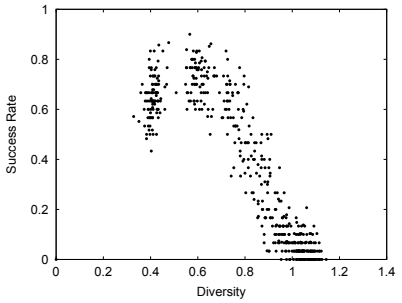
Figure 8: Average fitness error versus diversity after 100 substitute real-world generations

There appeared to be an optimum region of diversity that needed to be maintained to optimize the success of the proposed approach. The optimum diversity regions for Tasks 1 and 2 appeared to be similar (approximately between 0.5 to 0.7), whereas for Task 3 the optimum diversity was approximately between 0.4 to 0.6. These optimum diversity ranges appeared to correspond to where the fitness errors were consistently lower. An obvious reason for this would be that the simulator needed to be sufficiently accurate to evolve better controllers.

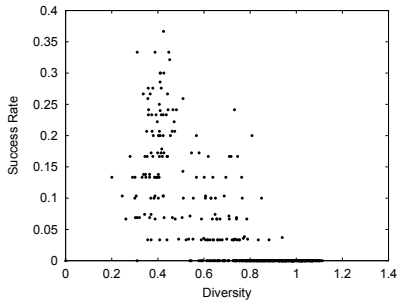
The Task 3 diversity versus success rate (Figure 9c) does not appear to have the pattern seen in Figures 9a and 9b. A possible reason may be the low success rates obtained for Task 3. The Task 3 diversity versus success rate figure was recalculated as shown in Figure 9d, with the goal point radius being slightly larger. This made it clear that all success versus diversity plots



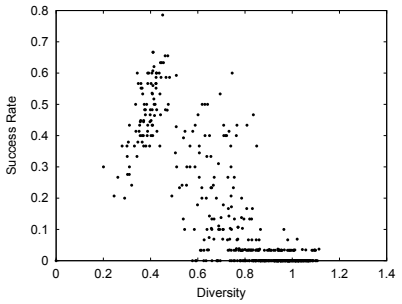
(a) Task 1 (using 7cm goal point radius)



(b) Task 2 (using 7cm goal point radius)



(c) Task 3 (using 7cm goal point radius)



(d) Task 3 (using 10cm goal point radius)

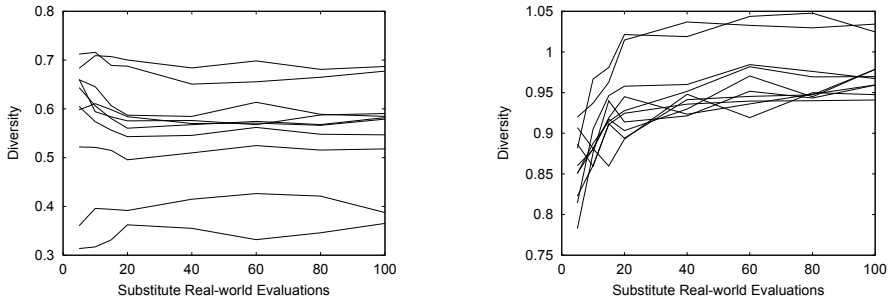
Figure 9: Success rate versus diversity after 100 substitute real-world generations

had a similar pattern. The best performing parameter combination for Task 1 (Figure 9a) had a success rate of about 76% and Task 2 (Figure 9b) had a success rate of about 90%. Task 3 had a poor success rate for the 7cm goal point radius, where success rates were below 37% (Figure 9c). If the goal point radius was relaxed to be 10cm, the best success rate improved to be roughly 79% (Figure 9d).

6.2.2. Top and Worst Performing Parameter Combinations

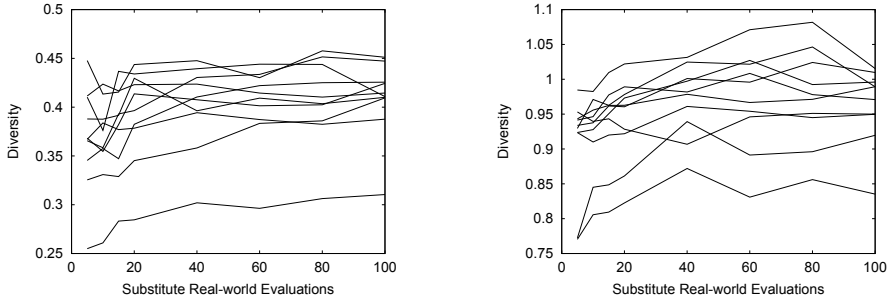
The best and worst performing parameter combinations were compared with each other in order to identify differences, thereby identifying the influential factors. The diversity over time for the top 10 best and worst performing parameter combinations for Tasks 1 and 3 can be seen in Figures 10 and 11 respectively. The Task 2 diversity over time was not included as a

figure due to its similarity to Task 1. The diversity for the first few substitute real-world evaluations of the proposed approach was usually the highest. However, there was little point in analysing how the diversity behaved before the fifth substitute real-world generation, as controller evaluations were almost random. For this reason, figures do not present the diversity before the fifth substitute real-world evaluation.



(a) Top 10 parameter combinations (b) Worst 10 parameter combinations

Figure 10: Diversity versus number of substitute real-world evaluations for Task 1



(a) Top 10 parameter combinations (b) Worst 10 parameter combinations

Figure 11: Diversity versus number of substitute real-world evaluations for Task 3

The top 10 best performing parameter combinations had a lower diversity when compared to the worst performers for each task. A higher diversity may have been due to controllers having a higher mutation rate which could

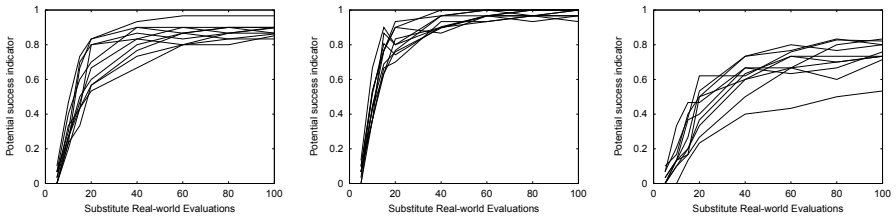
have resulted in controller evolution being unable to exploit good solutions. Controllers that are mutated too much, together with the issue of the SNNs continually changing may have contributed to higher mutation rates performing poorly. Most of the top 10 worst performers had mutation rates that were 70% or above for all tasks. The vast majority of the top 10 best performers had mutation rates of 30% or less.

In the previous section, the optimum diversity range was estimated based on success rates and fitness errors. Six of the ten ten for Task 2 and eight out of ten for Tasks 1 and 3 were within their estimated optimum ranges. The diversities of the best performers were fairly constant after the twentieth substitute real-world evaluation for the first two tasks and after the fortieth substitute real-world evaluation for the third task (Figures 10a and 11a). The diversity of the worst performers tended to increase over time for all tasks.

An interesting observation emerged where the first two tasks had a larger range of diversities for the best parameter combinations when compared to worst performers. The reverse was seen for Task 3, where the worst performers had a larger range of diversity. All worst performers for Task 3 had a 0% success rate for all levels of success. The ranking therefore had to be based on greatly varying fitness estimates which resulted in a wider diversity range. The higher diversities for worst performers indicate that a saturation point was reached where the diversity could not become much higher. These saturation ranges were confirmed by comparing them with the maximum diversities obtained in Figures 8 and 9. The top 10 best and worst parameter combinations for the other parameter types did not clearly show any trends, which may be due to the mutation rate being the dominating factor for success for top or worst performers.

The potential success indicator was important for determining how rapidly the desired behaviour was achieved and was a good indicator that successful solutions would be found. This indicator was defined to be the success rate for task completion when a robot traversed all goal points, but with the goal point radius being set to be twenty centimetres (for example, in Task 1, the robot would at least be moving through the correct quadrants of the xy-plane). The convergence towards the target behaviour for the top 10 best performing parameter combinations using the potential success indicator can be observed in Figure 12. A potential success indicator value of 50% was achieved by all top 10 best parameter combinations by the 20^{th}, 10^{th} and 100^{th} substitute real-world evaluation for Tasks 1, 2 and 3 respec-

tively. This indicated that Task 2 generally converged towards the desired behaviour the soonest, Task 1 converged slower and Task 3 converged much later. Task 3 clearly struggled to converge towards the desired behaviour which could indicate that solutions often converged to sub-optimal solutions or good solutions were frequently lost.

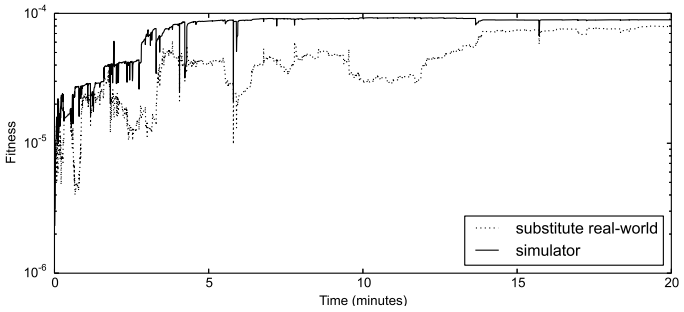


(a) Task 1 (b) Task 2 (c) Task 3

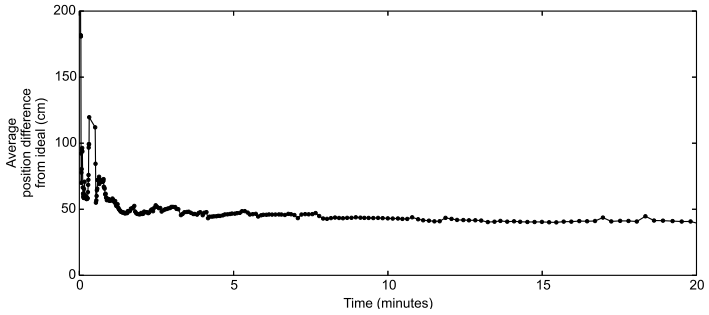
Figure 12: Potential success indicator versus number of substitute real world evaluations for the top 10 parameter combinations

Figure 13a illustrates the substitute real-world's observed fitnesses and the simulator's assigned fitnesses over the duration of a single experimental run of Task 2. The experiment terminated after thirty substitute real-world controller evaluations. The initial fitness assignments were volatile and inaccurate but over time stabilized and the simulator became more accurate. The substitute real-world fitness was mostly found to be less than the simulator's perceived fitness. A small drop in fitness was occasionally observed which corresponded to changes to the simulator.

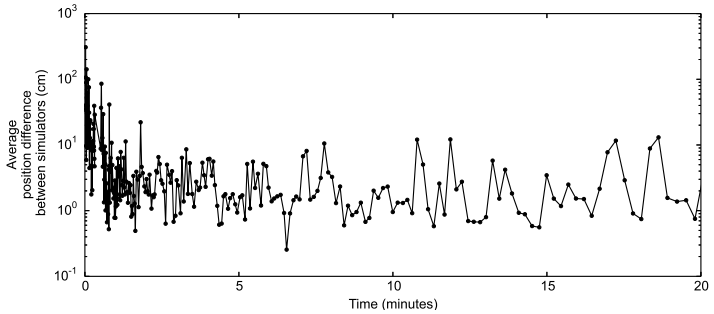
A set of ten randomly generated controllers were created before the experimental run and each controller contained five commands that were generated from a uniform distribution. These controllers were evaluated periodically over time and final positions reached were noted. Figure 13b represents how close these final positions were in simulation to the substitute real-world. The position difference varied significantly during the early stages of the experiment and later stabilized. Figure 13c presents the difference between the final positions of successive generations of the simulators. This demonstrated that there were large changes between successive simulators which gradually stabilized to become less severe.



(a) Fitness over time



(b) Position displacement over time



(c) Change in position displacement over time

Figure 13: Experimental run over time

7. CONCLUSIONS AND FUTURE WORK

A vital goal in ER worth exploring has been shown to be the automatic, real-time creation of controllers and simulators with minimal human intervention or specialized knowledge. SNNs have been shown to provide a viable alternative to physics-based simulators for this goal. Traditional SNN construction requires the collection of large amount of empirical data and may simulate unnecessary behaviours. Many traditional approaches to simulator development are complex, time-consuming and require specialized knowledge while SNNs are relatively simple to develop but can also be time-consuming. Bidirectional approaches to controller and simulator development have shown much promise in reducing the interaction times caused by real-world controller evaluations. The prototype developed in this paper demonstrated that a bidirectional approach to controller and SNN development during the ER process is indeed viable.

The proposed approach demonstrated accurate modelling of robot behaviour and also produced successful controllers for performing waypoint navigational tasks using a robot. This paper focused on developing simulators that were specialized according to the required behaviour. The specialization was due to the evaluation of task specific controllers for behavioural data, as opposed to the development of a more general simulator. The use of task specific controllers attempted to reduce the number of controller evaluations needed, thereby speeding up the ER process.

The controller population size and controller mutation rate were shown to be the most influential parameters in controlling the success of the proposed approach. However, there was no guarantee of a successful solution. Controller evolution may converge to sub-optimal solutions or good solutions may be lost due to a changing simulator. Higher population sizes can result in fewer instances where good solutions were lost due to a changing simulator. Lower mutation rates were shown to perform better than higher values which could be due to higher values not allowing controllers to exploit important features.

When compared to prior work the bidirectional approach proposed in this paper is likely to require fewer real-world evaluations than the traditional approach to SNN development. However, the choice between using a traditional or bidirectional SNN approach may be dependent on various factors. Traditional SNNs are focused on developing a generalized simulator that simulates a wide range of behaviours and is useful for evolving controllers for many dif-

ferent behaviours. Bidirectional SNNs are more focused on simulating only the behaviours required to evolve controllers for specific tasks.

Future work could investigate ways of improving the proposed approach. Other aspects regarding simulator training, controller evolution and automatic damage recovery could also be investigated. A future investigation into the viability of the proposed approach on a more complex robot is planned.

Acknowledgement

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged (UID number: 89526). Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

References

[1] J. C. Bongard, Evolutionary robotics, vol. 56, ACM, 2013.

[2] I. Harvey, P. Husbands, D. Cliff, Others, Issues in evolutionary robotics, School of Cognitive and Computing Sciences, University of Sussex, 1992.

[3] C. J. Pretorius, M. C. du Plessis, C. Cilliers, Simulating robots without conventional physics: A neural network approach, Journal of Intelligent & Robotic Systems 71 (3-4) (2013) 319–348.

[4] C. J. Pretorius, M. C. du Plessis, J. W. Gonsalves, A comparison of neural networks and physics models as motion simulators for simple robotic evolution, in: Evolutionary Computation (CEC), 2014 IEEE Congress on, IEEE, 2793–2800, 2014.

[5] J. C. Zagal, J. Ruiz-Del-Solar, Combining simulation and reality in evolutionary robotics, Journal of Intelligent and Robotic Systems 50 (1) (2007) 19–39, ISSN 0921-0296.

[6] N. Jakobi, P. Husbands, I. Harvey, Noise and the reality gap: The use of simulation in evolutionary robotics, in: Advances in artificial life, Springer, 704–720, 1995.

[7] D. Floreano, P. Husbands, S. Nolfi, Evolutionary robotics, Springer handbook of robotics (2008) 1423–1451.

[8] O. Miglino, K. Nafasi, C. Taylor, Selection for wandering behavior in a small robot, Artificial Life 2 (1) (1994) 101–116.

[9] M. Matarić, D. Cliff, Challenges in evolving controllers for physical robots, Robotics and autonomous systems 19 (1) (1996) 67–83.

[10] C. J. Pretorius, M. C. du Plessis, C. B. Cilliers, Towards an artificial neural network-based simulator for behavioural evolution in evolutionary robotics, in: Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, ACM, 170–178, 2009.

[11] D. Floreano, F. Mondada, Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot, From animals to animats (1994) 421–430.

[12] J. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, R. Watson, Evolutionary techniques in physical robotics, in: Evolvable Systems: from biology to hardware, Springer, 175–186, 2000.

[13] G. B. Parker, Co-evolving model parameters for anytime learning in evolutionary robotics, Robotics and Autonomous Systems 33 (1) (2000) 13–30.

[14] J. Bongard, V. Zykov, H. Lipson, Resilient machines through continuous self-modeling, Science 314 (5802) (2006) 1118–1121, ISSN 1095-9203.

[15] S. Koos, J. Mouret, S. Doncieux, The transferability approach: Crossing the reality gap in evolutionary robotics, Evolutionary Computation, IEEE Transactions on 17 (1) (2013) 122–145.

[16] A. Faiña, F. Bellas, F. Orjales, D. Souto, R. J. Duro, An evolution friendly modular architecture to produce feasible robots, Robotics and Autonomous Systems 63 (2015) 195–205.

[17] W. Lee, J. Hallam, H. Lund, A hybrid GP/GA approach for co-evolving controllers and robot bodies to achieve fitness-specified tasks, in: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on, IEEE, 384–389, 1996.

[18] H. Lipson, J. Pollack, Automatic design and manufacture of robotic lifeforms, Nature 406 (6799) (2000) 974–978.

[19] J. C. Zagal, J. Ruiz-del Solar, UCHILSIM: A dynamically and visually realistic simulator for the robocup four legged league, in: RoboCup 2004: Robot Soccer World Cup VIII, Springer, 34–45, 2005.

[20] R. De Nardi, O. E. Holland, Coevolutionary modelling of a miniature rotorcraft, Intelligent Autonomous Systems 10: IAS-10 (2008) 364.

[21] S. Nakamura, R. Saegusa, S. Hashimoto, A Hybrid Learning Strategy for Real Hardware of Swing-Up Pendulum, Journal of Advanced Computational Intelligence & Intelligent Informatics (JACIII) 11 (8).

[22] L. Wang, A hybrid genetic algorithm-neural network strategy for simulation optimization, Applied Mathematics and Computation 170 (2) (2005) 1329–1343, ISSN 00963003.

[23] K-Team, Khepera III, `http://www.k-team.com/mobile-robotics-products/khepera-iii`, accessed: November 2014, 2014.

[24] O. Miglino, H. Lund, S. Nolfi, Evolving mobile robots in simulated and real environments, Artificial life 2 (4) (1995) 417–434.

[25] J. C. Lee, Hacking the nintendo wii remote, Pervasive Computing, IEEE 7 (3) (2008) 39–45.

[26] Nintendo, Wii Official Site at Nintendo, `http://wii.com`, accessed: November 2014, 2014.

[27] C. J. Pretorius, Artificial Neural Networks as simulators for behavioural evolution in evolutionary robotics, Masters thesis, Nelson Mandela Metropolitan University, 2010.

[28] Heaton Research, Official Encog Site, `http://www.heatonresearch.com/encog`, accessed: December 2014, 2014.

[29] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: The RPROP algorithm, in: Neural Networks, 1993., IEEE International Conference on, IEEE, 586–591, 1993.

[30] A. P. Engelbrecht, Computational intelligence: an introduction, John Wiley & Sons, 2007.